Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы трансляции

ОТЧЁТ

к лабораторной работе №4

на тему

**СЕМАНТИЧЕСКИЙ АНАЛИЗ**

Выполнил: студент гр.253505 Таргонский Д.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2025

СОДЕРЖАНИЕ

Введение3

[1  [Ц](#_agswffz6g6ei)ель работы](#_Toc178067889) 4

[2  Теоретические сведения 5](#_Toc178067890)

[3  Описание и пример выполнения программы 7](#_Toc178067891)

Заключение9

[Список использованных источников 10](#_Toc178067900)

[Приложение А (обязательное) 11](#_Toc178067901)

# ВВЕДЕНИЕ

В процессе создания транслятора с PHP на VB .NET центральное значение приобретает этап семантического анализа, который следует за синтаксическим разбором исходного кода. На этой стадии проверяется «смысл» программы: анализируются объявленные типы, переменные и идентификаторы, порядок и корректность использования выражений и управляющих конструкций, чтобы убедиться, что они соответствуют семантике обоих языков. Цель семантического анализа — не только установить, что код «синтаксически» правильный, но и гарантировать его логическую непротиворечивость: отловить случаи использования необъявленных переменных, несоответствия типов при присваивании, неверного числа аргументов при вызове процедур и функций и другие ошибки, которые не фиксируются на этапе разбора.

При переносе программ из PHP в VB .NET дополнительную сложность создают различия в системе типов и правилах области видимости: динамическая типизация PHP требует преобразований к строго типизированному VB .NET, где каждая переменная должна быть явно объявлена и типизирована. Семантический анализатор в составе нашего транслятора отвечает за построение символических таблиц, проверку согласованности типов и генерацию корректных объявлений VB .NET-переменных на основе исходных PHP-конструкций.

Реализация семантики в среде .NET позволяет использовать сильную типизацию и богатый набор инструментов платформы — от встроенных коллекций для таблиц символов до механизма исключений для аккуратного сбора и представления обнаруженных ошибок. Такой подход делает транслятор не просто «конвертером текста», а полноценным средством контроля соответствия программ правилам целевого языка, приближая результат к стандартам промышленной разработки на VB .NET.

1 ЦЕЛЬ РАБОТЫ

Цель работы состоит в создании семантического анализатора для транслятора кода из PHP в VB .NET и исследовании его возможностей по проверке корректности программ на уровне семантики.

В процессе работы анализатор должен строить символические таблицы, аккумулировать сведения о типах и областях видимости переменных, функций и констант из PHP-кода, а затем последовательно проверять использование переменных до их объявления, соответствие типов при присваивании и в арифметических и логических выражениях, булевую корректность условий циклов и операторов ветвления, а также сопоставимость типов в конструкциях switch–case. При этом задача семантического анализатора не ограничивается обнаружением ошибок: он также обязан подготовить преобразования динамической типизации PHP в статически типизированные объявления VB .NET-переменных и, где необходимо, выполнить безопасные неявные приведения, сохраняя логику исходной программы.

Итогом работы станет единый механизм сбора и вывода семантических нарушений, обеспечивающий полное и понятное сообщение об ошибках, а также корректная генерация VB .NET-кода, отражающего оригинальные конструкции PHP с учётом требований целевого языка.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Семантический анализ — это важный этап в процессе компиляции программного кода, который проверяет смысловую корректность программы. В отличие от синтаксического анализа, который проверяет правильность структуры кода, семантический анализ фокусируется на логике и значениях, чтобы убедиться, что программа делает то, что задумано разработчиком. Это позволяет избежать множества ошибок, которые могут возникнуть при выполнении программы, и делает код более надежным и предсказуемым. [1]

Семантический анализ включает в себя множество проверок, таких как соответствие типов данных, корректность использования идентификаторов, проверка области видимости и корректность выполняемых операций. Эти проверки помогают разработчику убедиться, что программа работает так, как задумано, и позволяет избежать множества ошибок, которые могут возникнуть при выполнении программы.

Ппосле анализа синтаксиса программы следующим шагом является анализ семантики программы. Это включает в себя использование набора правил, чтобы определить, является ли программа семантически правильной, и создать промежуточное представление программы. Семантический анализ — это процесс использования набора правил для определения правильности программы с точки зрения семантики и создания промежуточного представления программы.

Цель семантического анализа двояка: во-первых, он проверяет, что все операции допустимы в соответствии с их определениями; во-вторых, он создает внутреннее представление, которое может использоваться другими частями компилятора или интерпретатора. Чтобы семантический анализ работал правильно, каждая операция должна иметь собственное определение в терминах того, какие входные данные она получает и какие выходные данные производит.

Результатом этого этапа должно быть абстрактное синтаксическое дерево (AST), содержащее информацию о том, как каждая операция работает в своем контексте. После создания этого AST может выполняться дальнейшая обработка, например проверка типов или оптимизация. Семантический анализ помогает убедиться, что программы синтаксически корректны до их выполнения, что позволяет обнаруживать ошибки на ранних этапах разработки. [2]

Семантический анализ выполняет несколько ключевых задач, каждая из которых направлена на обеспечение корректности и надежности кода

Одна из основных задач семантического анализа — проверка соответствия типов данных. Например, если вы пытаетесь сложить число и строку, компилятор должен выдать ошибку. Это предотвращает множество ошибок, связанных с некорректным использованием типов данных. Например, в языке Python попытка сложить число и строку приведет к ошибке времени выполнения, тогда как компилятор языка C++ может выявить такую ошибку на этапе компиляции.

Семантический анализ также проверяет, что все переменные и функции объявлены перед использованием. Это помогает избежать ошибок, связанных с неопределенными идентификаторами. Например, если вы попытаетесь использовать переменную, которая не была объявлена, компилятор выдаст ошибку. Это позволяет избежать множества ошибок, связанных с неправильным использованием идентификаторов.

Компилятор проверяет, что переменные и функции используются в пределах своей области видимости. Это предотвращает ошибки, связанные с доступом к переменным из других блоков кода. Например, если вы попытаетесь использовать переменную, объявленную внутри функции, за пределами этой функции, компилятор выдаст ошибку. Это помогает избежать множества ошибок, связанных с неправильным использованием областей видимости.

Семантический анализ также проверяет, что операции выполняются над корректными типами данных. Например, деление на ноль или использование неинициализированных переменных. Это позволяет избежать множества ошибок, связанных с некорректными операциями. Например, в языке Python попытка деления на ноль приведет к ошибке времени выполнения, тогда как компилятор языка C++ может выявить такую ошибку на этапе компиляции.

Семантический анализатор использует синтаксическое дерево и информацию из таблицы идентификаторов для проверки входной программы на семантическую согласованность с определением языка программирования. Он также собирает информацию о типах и сохраняет её в синтаксическом дереве или в таблице идентификаторов для последующего использования в процессе генерации промежуточного кода. [3]

После синтаксического и семантического анализа исходной программы компиляторы генерируют низкоуровневое промежуточное представление входной программы, которое можно рассматривать как программу для абстрактной вычислительной машины. Такое промежуточное представление должно обладать двумя важными свойствами: оно должно легко генерироваться и легко транслироваться в целевой машинный язык.

Семантика в программировании — это фундаментальное понятие, которое помогает разработчикам понимать и предсказывать поведение программного кода. Понимание семантики позволяет писать более надежные и предсказуемые программы. Для дальнейшего изучения рекомендуется ознакомиться с книгами и статьями по теории языков программирования, а также практиковаться в написании кода на различных языках программирования.

Изучение семантики также помогает лучше понять, как работают компиляторы и интерпретаторы. Это знание может быть полезным при оптимизации кода и создании более эффективных программ. Кроме того, понимание семантики помогает лучше понимать документацию и примеры кода, что делает процесс обучения новым языкам программирования более быстрым и эффективным.

**3 ОПИСАНИЕ И ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Разработанный в рамках проекта семантический анализатор представляет собой ключевую часть транслятора кода из PHP в VB .NET. Он принимает на вход уже разбитый на токены и проверенный синтаксисом синтаксический буфер и выполняет всесторонний смысловой анализ: строит символические таблицы, фиксирует информацию о типах и областях видимости переменных и функций, контролирует корректность операций присваивания и математических выражений, проверяет булевость условий в if и заголовках циклов, сопоставляет типы в конструкциях switch–case. При нахождении нарушений — например, попыток присвоить строку переменной, ранее объявленной как целочисленная, использования переменных до их объявления или применении арифметических операторов к несовместимым типам — анализатор формирует чёткие диагностические сообщения с указанием строк кода и описанием ошибок.

В ходе тестирования семантический анализатор был проверен на нескольких демонстрационных фрагментах PHP-кода. Первый фрагмент (рисунок 3.1) содержал четыре целенаправленно введённые ошибки: присваивание строкового значения целочисленной переменной $a, использование несуществующей переменной $b в выражении, некорректное применение оператора + и некорректное условие if (123) вместо булева. Анализатор успешно обнаружил и отобразил все эти нарушения, что наглядно демонстрирует его способность обеспечивать логическую непротиворечивость исходного кода перед этапом генерации VB .NET.

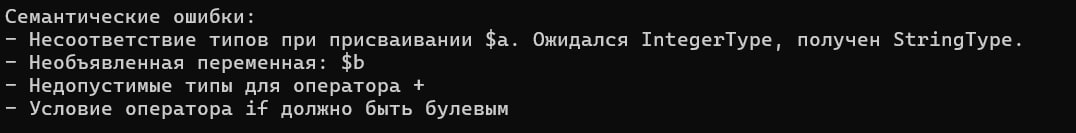


Рисунок 3.1 – Результат семантического анализа на программе с предмаеренными семантическими ошибками

Во втором тесте (рисунок 3.2) был проанализирован корректный фрагмент PHP-кода. Все переменные и массивы были объявлены до первого использования, типы присваиваний и условий соответствовали требованиям целевого языка, а зарезервированные слова не применялись как идентификаторы. В результате семантический анализ завершился без единого сообщения об ошибке.

Изображение выглядит как Шрифт, текст, снимок экрана, Графика

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.2 – Результат второго теста

Таким образом, разработанный анализатор надёжно выявляет самые разные смысловые нарушения на этапе подготовки PHP-кода к переводам в VB .NET, предоставляя исчерпывающую диагностическую информацию, необходимую для оперативного исправления исходных программ.

# ЗАКЛЮЧЕНИЕ

В ходе данной работы был разработан семантический анализатор для транслятора, преобразующего код PHP в VB.NET. Основной задачей анализатора являлась проверка корректности семантики исходного PHP-кода и обеспечение его корректного соответствия правилам языка VB.NET. В процессе разработки были изучены особенности обоих языков, включая их синтаксические конструкции, типизацию, правила видимости переменных и функций, а также механизмы работы с объектами и массивами.

Для реализации семантического анализатора были применены методы статического анализа кода, включая построение абстрактного синтаксического дерева (AST), проверку типов, контроль областей видимости и валидацию вызовов функций. Особое внимание уделялось обработке динамических особенностей PHP, таких как переменные переменные и ассоциативные массивы, которые требуют дополнительных преобразований при переводе в строго типизированный VB.NET.

Разработанный анализатор успешно интегрирован в общую архитектуру транслятора и позволяет выявлять семантические ошибки на ранних этапах преобразования кода. Это значительно снижает количество потенциальных проблем при генерации итогового VB.NET-кода и упрощает процесс отладки.

Проделанная работа подтвердила возможность автоматизированного перевода PHP-кода в VB.NET с сохранением функциональности, а также выявила ряд направлений для дальнейшего улучшения, таких как оптимизация обработки сложных динамических конструкций и расширение поддерживаемого подмножества PHP. В перспективе данный инструмент может быть доработан для поддержки более широкого спектра языковых возможностей и интеграции в современные среды разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Основы построения трансляторов языков программирования. Е. В. Шостак, И. М. Марина, Д. Е. Оношко [Электронный ресурс]. – Режим доступа: https://libeldoc.bsuir.by/bitstram/123456789/35077/1/Shostak\_2019.pdf.

[2] Компиляторы: принципы, методы и инструменты. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульманрёв [Электронный ресурс]. – Режим доступа: https://vsuhom.ru/pdf/ru/book/www.vsuhom.ru\_744\_abstrakt-Ko mpilyatory\_princip.pdf.

[3] Кнспектное изложение теории языков программирования и методов трансляции. Вл. Понаморёв [Электронный ресурс]. – Режим доступа: https://t imeweb.cloud/tutorials/linux/regulyarnye—vyrazheniya—bash—gajd?ysclid=m82 zxlerds129168253.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

Imports System.Collections.Generic

Public Enum DataType

IntegerType

DoubleType

StringType

BooleanType

ArrayType

ObjectType

Unknown

End Enum

Public Class Symbol

Public Property Name As String

Public Property Type As DataType

Public Sub New(name As String, type As DataType)

Me.Name = name

Me.Type = type

End Sub

End Class

Public Class SymbolTable

Private symbols As New Dictionary(Of String, Symbol)

Public Sub AddSymbol(name As String, type As DataType)

If symbols.ContainsKey(name) Then

Throw New Exception("Переменная " & name & " уже объявлена в этой области видимости.")

End If

symbols.Add(name, New Symbol(name, type))

End Sub

Public Function GetSymbol(name As String) As Symbol

If symbols.ContainsKey(name) Then

Return symbols(name)

End If

Return Nothing

End Function

End Class

Public Class SemanticAnalyzer

Private symbolTables As New Stack(Of SymbolTable)

Private currentTable As SymbolTable

Private errors As New List(Of String)

Public Sub New()

currentTable = New SymbolTable()

symbolTables.Push(currentTable)

End Sub

''' <summary>

''' Анализирует AST и возвращает список семантических ошибок.

''' </summary>

Public Function Analyze(syntaxTree As ASTNode) As List(Of String)

errors.Clear()

AnalyzeNode(syntaxTree)

Return errors

End Function

Private Sub ReportError(message As String)

errors.Add(message)

End Sub

Private Sub AnalyzeNode(node As ASTNode)

If node Is Nothing Then Return

Select Case node.NodeType

Case "Program", "Block", "Statement"

For Each ch In node.Children

AnalyzeNode(ch)

Next

Case "Assignment"

Try

AnalyzeAssignment(node)

Catch ex As Exception

ReportError(ex.Message)

End Try

Case "BinaryExpression"

' Спец-обработка присваивания в заголовке for

If node.Value = "=" AndAlso node.Children(0).NodeType = "Identifier" Then

Try

Dim id = node.Children(0).Value

Dim expr = node.Children(1)

Dim t = GetExpressionType(expr)

Dim sym = GetSymbol(id)

If sym Is Nothing Then

currentTable.AddSymbol(id, t)

ElseIf sym.Type <> t Then

ReportError($"Несоответствие типов при присваивании {id}. Ожидался {sym.Type}, получен {t}.")

End If

Catch ex As Exception

ReportError(ex.Message)

End Try

Else

GetExpressionType(node)

End If

Case "ExpressionStatement"

GetExpressionType(node.Children(0))

Case "ForStatement"

' Инициализация, условие, инкремент, тело

AnalyzeNode(node.Children(0).Children(0))

AnalyzeNode(node.Children(1).Children(0))

AnalyzeNode(node.Children(2).Children(0))

AnalyzeNode(node.Children(3).Children(0))

Case "WhileStatement"

AnalyzeNode(node.Children(0).Children(0))

AnalyzeNode(node.Children(1).Children(0))

Case "DoWhileStatement"

AnalyzeNode(node.Children(0).Children(0))

AnalyzeNode(node.Children(1).Children(0))

Case "IfStatement"

AnalyzeNode(node.Children(0).Children(0))

AnalyzeNode(node.Children(1).Children(0))

If node.Children.Count > 2 Then AnalyzeNode(node.Children(2).Children(0))

Case "SwitchStatement"

AnalyzeNode(node.Children(0).Children(0))

For i As Integer = 1 To node.Children.Count - 1

For Each ch In node.Children(i).Children

AnalyzeNode(ch)

Next

Next

Case "FunctionDeclaration"

' Новый scope для функции

Dim fnTable = New SymbolTable()

symbolTables.Push(fnTable)

currentTable = fnTable

For Each p In node.Children(1).Children

currentTable.AddSymbol(p.Value, DataType.Unknown)

Next

AnalyzeNode(node.Children(2).Children(0))

symbolTables.Pop()

currentTable = symbolTables.Peek()

Case "EchoStatement"

For Each arg In node.Children(0).Children

GetExpressionType(arg)

Next

Case "UnaryExpression"

GetExpressionType(node)

End Select

End Sub

Private Sub AnalyzeAssignment(node As ASTNode)

Dim id = node.Children(0).Value

Dim exprType = GetExpressionType(node.Children(1))

Dim sym = GetSymbol(id)

If sym Is Nothing Then

currentTable.AddSymbol(id, exprType)

ElseIf sym.Type <> exprType Then

ReportError($"Несоответствие типов при присваивании {id}. Ожидался {sym.Type}, получен {exprType}.")

End If

End Sub

Private Function GetExpressionType(expr As ASTNode) As DataType

Select Case expr.NodeType

Case "Literal"

If IsNumeric(expr.Value) Then

Return If(expr.Value.Contains("."), DataType.DoubleType, DataType.IntegerType)

ElseIf expr.Value.ToLower() = "true" OrElse expr.Value.ToLower() = "false" Then

Return DataType.BooleanType

Else

Return DataType.StringType

End If

Case "Identifier"

Dim s = GetSymbol(expr.Value)

If s Is Nothing Then

ReportError($"Необъявленная переменная: {expr.Value}")

Return DataType.Unknown

End If

Return s.Type

Case "BinaryExpression"

Dim lt = GetExpressionType(expr.Children(0))

Dim rt = GetExpressionType(expr.Children(1))

Select Case expr.Value

Case "+", "-", "\*", "/"

If lt = rt AndAlso lt = DataType.IntegerType Then Return DataType.IntegerType

If (lt = DataType.IntegerType OrElse lt = DataType.DoubleType) AndAlso

(rt = DataType.IntegerType OrElse rt = DataType.DoubleType) Then

Return DataType.DoubleType

End If

ReportError($"Недопустимые типы для оператора {expr.Value}")

Return DataType.Unknown

Case "==", "!=", "<", ">", "<=", ">="

If lt = rt Then Return DataType.BooleanType

ReportError("Несоответствие типов в сравнении")

Return DataType.Unknown

End Select

Return DataType.Unknown

Case "UnaryExpression"

Dim ot = GetExpressionType(expr.Children(0))

If expr.Value = "++" OrElse expr.Value = "--" Then

If ot = DataType.IntegerType OrElse ot = DataType.DoubleType Then

Return ot

End If

ReportError($"Недопустимый тип для унарного оператора {expr.Value}")

End If

Return DataType.Unknown

Case Else

Return DataType.Unknown

End Select

End Function

Private Function GetSymbol(name As String) As Symbol

For Each tbl In symbolTables

Dim s = tbl.GetSymbol(name)

If s IsNot Nothing Then Return s

Next

Return Nothing

End Function

End Class

?>