Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы трансляции

ОТЧЁТ
по лабораторной работе
на тему

Синтаксический анализ

Выполнил
Студент гр. 053501
Хиль В.М.

Проверил
Ассистент кафедры информатики
Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Освоение работы с существующими синтаксическими анализаторами. Разработать свой собственный синтаксический анализатор, выбранного подмножества языка программирования. Построить синтаксическое дерево. Определить минимум 4 возможных синтаксических ошибки и показать их корректное выявление.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В ходе синтаксического анализа исходный текст программы проверяется на на соответствие синтаксическим нормам языка с построением дерево разбора (синтаксическое дерево), которое отражает синтаксическую структуру входной последовательности и удобно для дальнейшего использования, а также в случае несоответствия – позволяет вывести сообщения об ошибках.

Как правило, результатом синтаксического анализа является синтаксическое строение предложения, представленное либо в виде дерева зависимостей, либо в виде дерева составляющих, либо в виде некоторого сочетания первого и второго способов представления.

Таким образом на основе анализа выражений, состоящих из литералов, операторов и круглых скобок выполняется группирование токенов исходной программы в грамматические фразы, используемые для синтеза вывода. Представление грамматических фраз исходной программы выполнить в виде дерева. Реализовать синтаксический анализатор с использованием одного из табличных методов (LL-, LR-метод, метод предшествования и пр.).

Т.е. мы имеем последовательность шагов в виде помеченного дерева. Внутренние вершины представляют те действия, которые можно выполнять. Прямые потомки каждой вершины либо представляют аргументы, к которым нужно применять действие (если соответствующая вершина помечена идентификатором или является внутренней), либо помогают определить, каким должно быть это действие, в частности, знаки «+», «*» и «=». Скобки отсутствуют, т.к. они только определяют порядок действий.

LL- и LR- методы позволят обнаружить ошибки на самых ранних стадиях, т.е. когда разбор потока токенов от лексического анализатора в соответствии с грамматикой языка становится невозможен. Можно использовать нисходящий (англ. top-down parser) со стартового символа, до получения требуемой последовательности токенов. Для этих целей применим метод рекурсивного спуска либо LL-анализатор. Или использовать восходящий (англ. bottom-up parser) - продукции восстанавливаются из правых частей, начиная с токенов и кончая стартовым символом - LR-анализатор и прочее.

3 ПРИМЕРЫ РАБОТЫ СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА

Текущая версия программы передает список токенов и файл, содержащий грамматику синтаксическому анализатору, который согласно правилам, описанным там, с помощью LR-алгоритма формирует синтаксическое дерево.

3.1 Работа анализатора

Для примера возьмем следующую программу (см. рисунок 1):

```
FUN fact(n)
    VAR result = 1
    IF n < 0 THEN
        PRINT(result)
    END
END

fact(-1)
```

Рисунок 1 – Входные данные

В результате было построено синтаксическое дерево (см. рисунок 2), состоящее из различных синтаксических узлов программы (см. приложение Б):

```
ListNodes:[
--FuncDefNode:
----VarName:IDENTIFIER:fact,
------ArgNameFunc:[IDENTIFIER:n],
--------Body:
ListNodes:[
----------VarAssignNode:
------------TypeVar:IDENTIFIER:result,
--------------Value:
----------------Type:INT,
------------------Value:1,
----------IfNode:
------------Cases:[
--------------
----------------BinOpNode:
------------------LeftNode:
--------------------VarAccessNode:
----------------------IDENTIFIER:n,
--------------------Operation:LT,
----------------------RightNode:
------------------------Type:INT,
--------------------------Value:0,
ListNodes:[
----------------CallNode:
------------------NodeToCall:
--------------------VarAccessNode:
----------------------IDENTIFIER:PRINT,
--------------------ArgNodeToCall:[
----------------------VarAccessNode:
------------------------IDENTIFIER:result]], True],
--------------Else:None],
--CallNode:
----NodeToCall:
------VarAccessNode:
--------IDENTIFIER:fact,
------ArgNodeToCall:[
--------UnaryOpNode:
----------Operation:MINUS,
------------NodeUnary:
--------------Type:INT,
----------------Value:1]]
```

Рисунок 2 – Результат работы синтаксического анализатора

3.2 Синтаксические ошибки

Синтаксические ошибки возникают при выполнении синтаксического анализа и обычно указывают на то, что в коде есть неправильно написанные или расставленные элементы. Чтобы исправить синтаксические ошибки, нужно провести проверку на то, что все элементы расставлены правильно и соответствуют правилам выбранного подмножества языка.

Ошибка нарушения баланса скобок представлена на рисунке 3.

```
FUN fact(((n)
    VAR result = 1
    IF n < 0 THEN
```

Рисунок 3 – Программа с ошибкой

На рисунке 4 приведен результат работы синтаксического анализатора. Он выдал ошибку и указал, что не так.

```
Invalid Syntax: Expected identifier or ')'
File <stdin>, line 2


FUN fact(((n)
         ^
```

Рисунок 4 – Результат работы синтаксического анализатора

Ошибка неполного объявления переменной представлена на рисунке 5.

```
FUN fact(n)
    VAR  = 1
    IF n < 0 THEN
```

Рисунок 5 – Программа с ошибкой

На рисунке 6 приведен результат работы синтаксического анализатора. Он выдал ошибку и указал, что не так.

```
Invalid Syntax: Expected identifier
File <stdin>, line 3


VAR  = 1
      ^
```

Рисунок 6 – Результат работы синтаксического анализатора

Ошибка неожиданного токена представлена на рисунке 7.

```
FUN fact(n)
    VAR a ba = 1
    IF n < 0 THEN
```

Рисунок 7 – Программа с ошибкой

На рисунке 8 приведен результат работы синтаксического анализатора. Он выдал ошибку и указал, что не так.

```
Invalid Syntax: Expected '='
File <stdin>, line 3


VAR a ba = 1
       ^^
```

Рисунок 8 – Результат работы синтаксического анализатора

Ошибка неправильной последовательности токенов представлена на рисунке 9.

```
next <= n THEN
    PRINT(next)
    VAR el1 = el2
    VAR el2 = next
    VAR next = el1 + el2
END
```

Рисунок 9 – Программа с ошибкой

На рисунке 10 приведен результат работы синтаксического анализатора. Он выдал ошибку и указал, что не так.

```
Invalid Syntax: token cannot appear after previous tokens
File <stdin>, line 2


 next <= n THEN
           ^^^^
```

Рисунок 10 – Результат работы синтаксического анализатора

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

```
class Parser:
    def __init__(self, tokens):
        self.tokens = tokens
        self.token_index = -1
        self.advance()

    def update_current_token(self):
        if 0 <= self.token_index < len(self.tokens):
            self.current_token = self.tokens[self.token_index]

    def advance(self):
        self.token_index += 1
        self.update_current_token()
        return self.current_token

    def reverse(self, amount=1):
        self.token_index -= amount
        self.update_current_token()
        return self.current_token

    def parse(self):
        result = self.statements()
        if not result.error and self.current_token.type != EOF:
            return result.failure(InvalidSyntaxError(
                self.current_token.start_position,
self.current_token.end_position,
                "token cannot appear after previous tokens"
            ))
        return result

class ParseResult:
    def __init__(self):
        self.error = None
        self.node = None
        self.last_registered_count_advance = 0
        self.count_advance = 0
        self.to_reverse_count = 0

    def register(self, result):
        self.last_registered_count_advance = result.count_advance
        self.count_advance += result.count_advance
        if result.error: self.error = result.error
        return result.node

    def try_to_register(self, result):
        if result.error:
            self.to_reverse_count = result.count_advance
            return None
        return self.register(result)

    def success(self, node):
        self.node = node
        return self

    def failure(self, error):
        if not self.error or self.last_registered_count_advance == 0:
            self.error = error
```

```
        return self

    def register_advancement(self):
        self.last_registered_count_advance = 1
        self.count_advance += 1
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Узлы дерева

```
class ListNode:
    def __init__(self, element_nodes, start_position, end_position):
        self.element_nodes = element_nodes

        self.start_position = start_position
        self.end_position = end_position

    def __repr__(self):
        return f'\n\033[91mListNodes:{self.element_nodes}\033[91m'


class NumberNode:
    def __init__(self, token):
        self.token = token

        self.start_position = self.token.start_position
        self.end_position = self.token.end_position

    def __repr__(self):
        return f'(\033[31mType:{self.token.type},
(\033[31mValue:{self.token.value}\033[31m)\033[31m)'


class StringNode:
    def __init__(self, token):
        self.token = token

        self.start_position = self.token.start_position
        self.end_position = self.token.end_position

    def __repr__(self):
        return f'(\033[31mType:{self.token.type},
(\033[31mValue:{self.token.value}\033[31m)\033[31m)'


class BinaryOperationNode:
    def __init__(self, left_node, operation_token, right_node):
        self.left_node = left_node
        self.operation_token = operation_token
        self.right_node = right_node

        self.start_position = self.left_node.start_position
        self.end_position = self.right_node.end_position

    def __repr__(self):
        return f'(\033[35mBinOpNode:(\033[35mLeftNode:{self.left_node},
(\033[35mOperation:{self.operation_token},' \

f'(\033[35mRightNode:{self.right_node}\033[35m)\033[35m)\033[35m)\033[35m)'


class UnaryOperationNode:
    def __init__(self, operation_token, node):
        self.operation_token = operation_token
        self.node = node

        self.start_position = self.operation_token.start_position
```

```
        self.end_position = node.end_position

    def __repr__(self):
        return
f'(\033[35mUnaryOpNode:(\033[35mOperation:{self.operation_token},
(\033[35mNodeUnary:{self.node}' \
               f'\033[35m)\033[35m)\033[35m)'


class VarAccessNode:
    def __init__(self, var_name_token):
        self.var_name_token = var_name_token

        self.start_position = self.var_name_token.start_position
        self.end_position = self.var_name_token.end_position

    def __repr__(self):
        return
f'(\033[33mVarAccessNode:(\033[33m{self.var_name_token}\033[33m)\033[33m)'


class VarAssignNode:
    def __init__(self, var_name_token, value_node):
        self.var_name_token = var_name_token
        self.value_node = value_node

        self.start_position = self.var_name_token.start_position
        self.end_position = self.value_node.end_position

    def __repr__(self):
        return
f'(\033[34mVarAssignNode:(\033[34mTypeVar:{self.var_name_token},
(\033[34mValue:{self.value_node}' \
               f'\033[34m)\033[34m)\033[34m)'


class IfNode:
    def __init__(self, cases, else_case):
        self.cases = cases
        self.else_case = else_case

        self.start_position = self.cases[0][0].start_position
        self.end_position = (self.else_case or self.cases[len(self.cases) -
1])[0].end_position

    def __repr__(self):
        return f'(\033[36mIfNode:(\033[36mCases:{self.cases},
(\033[36mElse:{self.else_case}\033[36m)\033[36m)\033[36m)'


class ForNode:
    def __init__(self, var_name_token, start_value_node, end_value_node,
step_value_node, body_node, should_return_null):
        self.var_name_token = var_name_token
        self.start_value_node = start_value_node
        self.end_value_node = end_value_node
        self.step_value_node = step_value_node
        self.body_node = body_node
        self.should_return_null = should_return_null

        self.start_position = self.var_name_token.start_position
        self.end_position = self.body_node.end_position
```

```
    def __repr__(self):
        return f'(\033[37mForNode:(\033[37mVarName:{self.var_name_token}, ' \
               f'(\033[37mStartValue:{self.start_value_node}, ' \
               f'(\033[37mEndValue:{self.end_value_node}, ' \
               f'(\033[37mStep:{self.step_value_node},
(\033[37mBodyFor:{self.body_node}\033[37m)\033[37m)\033[37m)' \
               f'\033[37m)\033[37m)\033[37m)'


class WhileNode:
    def __init__(self, condition_node, body_node, should_return_null):
        self.condition_node = condition_node
        self.body_node = body_node
        self.should_return_null = should_return_null

        self.start_position = self.condition_node.start_position
        self.end_position = self.body_node.end_position

    def __repr__(self):
        return
f'(\033[37mWhileNode:(\033[37mConditionWhile:{self.condition_node}, ' \

f'(\033[37mBodyWhile:{self.body_node}\033[37m)\033[37m)\033[37m)'


class ContinueNode:
    def __init__(self, start_position, end_position):
        self.start_position = start_position
        self.end_position = end_position


class BreakNode:
    def __init__(self, start_position, end_position):
        self.start_position = start_position
        self.end_position = end_position


class FuncDefinitionNode:
    def __init__(self, var_name_token, arguments_name_tokens, body_node,
should_auto_return):
        self.var_name_token = var_name_token
        self.arguments_name_tokens = arguments_name_tokens
        self.body_node = body_node
        self.should_auto_return = should_auto_return

        if self.var_name_token:
            self.start_position = self.var_name_token.start_position
        elif len(self.arguments_name_tokens) > 0:
            self.start_position =
self.arguments_name_tokens[0].start_position
        else:
            self.start_position = self.body_node.start_position

        self.end_position = self.body_node.end_position

    def __repr__(self):
        return f'(\033[0mFuncDefNode:(\033[0mVarName:{self.var_name_token}, '
\
               f'(\033[0mArgNameFunc:{self.arguments_name_tokens}, ' \

f'(\033[0mBody:{self.body_node}\033[0m)\033[0m)\033[0m)\033[0m)'
```

```
class CallNode:
    def __init__(self, node_to_call, arguments_nodes):
        self.node_to_call = node_to_call
        self.arguments_nodes = arguments_nodes

        self.start_position = self.node_to_call.start_position

        if len(self.arguments_nodes) > 0:
            self.end_position =
self.arguments_nodes[len(self.arguments_nodes) - 1].end_position
        else:
            self.end_position = self.node_to_call.end_position

    def __repr__(self):
        return f'(\033[32mCallNode:(\033[32mNodeToCall:{self.node_to_call}, '
\

f'(\033[32mArgNodeToCall:{self.arguments_nodes}\033[32m)\033[32m)\033[32m)'


class ReturnNode:
    def __init__(self, node_to_return, start_position, end_position):
        self.node_to_return = node_to_return

        self.start_position = start_position
        self.end_position = end_position

    def __repr__(self):
        return
f'(\033[32mReturnNode:(\033[32mReturn:{self.node_to_return}\033[32m)\033[32m)
'
```

ПРИЛОЖЕНИЕ В

(обязательное)

Классы для обнаружения ошибок

```
from strings_with_error import arrow_string


class Error:
    def __init__(self, start_position, end_position, error_name, details):
        self.start_position = start_position
        self.end_position = end_position
        self.error_name = error_name
        self.details = details

    def string_representation(self):
        result = f'{self.error_name}: {self.details}\n'
        result += f'File {self.start_position.function}, line
{self.start_position.line + 1}'
        result += '\n\n' + arrow_string(self.start_position.function_text,
self.start_position, self.end_position)
        return result


class IllegalCharError(Error):
    def __init__(self, start_position, end_position, details):
        super().__init__(start_position, end_position, 'Illegal Character',
details)


class ExpectedCharError(Error):
    def __init__(self, start_position, end_position, details):
        super().__init__(start_position, end_position, 'Expected Character',
details)


class InvalidSyntaxError(Error):
    def __init__(self, start_position, end_position, details=''):
        super().__init__(start_position, end_position, 'Invalid Syntax',
details)
```