Университет ИТМО
Кафедра ИПМ

Верификация моделей программ
Лабораторная работа 1
«Построение синтаксического дерева»
Вариант - 13

Выполнил:
Морозов С.Д.
группа P4217
Преподаватель:
Кореньков Ю.Д.

Санкт-Петербург
2019

Цель: освоение верификации синтаксических конструкций текстовых представлений данных.

Задачи: Реализовать построение по исходному файлу с текстом синтаксического дерева с узлами, соответствующими правилам варианта, задающего язык для анализа. Вывести полученное дерево в файл.

Вариант:

Типизация | Байт-код | Параметризация
Статическая| Регистровый |Шаблонизация

Входной язык:

```
identifier: "[a-zA-Z_][a-zA-Z_0-9]*";   // идентификатор
str: "\"[^\"\\]*(?:\\.[^\"\\]*)*\"";    // строка, окруженная двойными кавычками
char: "'[^']'";                         // одиночный символ в одинарных кавычках
hex: "0[xX][0-9A-Fa-f]+";               // шестнадцатеричный литерал
bits: "0[bB][01]+";                     // битовый литерал
dec: "[0-9]+";                          // десятичный литерал
bool: 'true'|'false';                   // булевский литерал
list<item>: (item (',' item)*)?;        // список элементов, разделённых запятыми

source: sourceItem*;

typeRef: {
 |builtin: 'bool'|'byte'|'int'|'uint'|'long'|'ulong'|'char'|'string';
 |custom: identifier;
 |array: typeRef '(' (',')* ')';

};

funcSignature: identifier '(' list<argDef> ')' ('as' typeRef)? {
 argDef: identifier ('as' typeRef)?;
};

source: sourceItem*;

sourceItem: {
 |funcDef: 'function' funcSignature statement* 'end' 'function';
```

```
};

statement: {
 |var: 'dim' list<identifier> 'as' typeRef;// for static typing
 |if: 'if' expr 'then' statement* ('else' statement*)? 'end' 'if';
 |while: 'while' expr statement* 'wend';
 |do: 'do' statement* 'loop' ('while'|'until') expr;
 |break: 'break';
 |expression: expr ';';
 };

expr: { // присваивание через '='
 |binary: expr binOp expr; // где binOp - символ бинарного оператора
 |unary: unOp expr; // где unOp - символ унарного оператора
 |braces: '(' expr ')';
 |callOrIndexer: expr '(' list<expr> ')';
 |place: identifier;
 |literal: bool|str|char|hex|bits|dec;
};
```

Описание работы

Программа по построению синтаксического дерева реализована на языке программирования Python с использованием библиотеки ply (http://www.dabeaz.com/ply/ply.html#ply_nn24) для разбиения входного теста на токены (лексический анализ, ply.lex) и для преобразования токенов в синтаксическое дерево (ply.yacc). Для построения дерева использовалась библиотека anytree (https://anytree.readthedocs.io/en/latest/index.html) т.к. данная библиотека обладает удобным преобразованием дерева в текстовое представление и имеет возможность экспорта дерева в dot файл, который в последствии можно преобразовать в графическое изображение дерева используя (https://www.graphviz.org/)

Аспекты работы:

Работу можно разделить на 2 основные части

1) Лексический анализ
2) Построение синтаксического дерева

Лексический анализ:

Были выбраны следующие типы токенов:

Типы переменных

'BUILTIN_BOOL', 'BUILTIN_BYTE', 'BUILTIN_INT', 'BUILTIN_UINT', 'BUILTIN_LONG', 'BUILTIN_ULONG', 'BUILTIN_CHAR', 'BUILTIN_STRING', 'BUILTIN_LIST',

Зарезервированные конструкции языка

'AS', 'FUNCTION', 'END', 'IF', 'THEN', 'ELSE', 'WHILE', 'WEND', 'DO', 'LOOP', 'UNTIL', 'BREAK', 'COMMA', 'COLON', 'SEMICOLON', 'DIM',

Операции

'PLUS', 'MINUS', 'DIVIDE', 'MUL', 'LESS_EQ', 'LESS', 'MORE_EQ', 'MORE', 'NOT_EQ', 'EQUAL', 'AND', 'OR', 'NOT', 'ASSIGNMENT', 'LBRACES', 'RBRACES',

Переменные

'BOOL', 'BYTE', 'DEC', 'HEX', 'BITS', 'INT', 'UINT', 'LONG', 'ULONG', 'CHAR', 'STR',

Имена переменных

‘'IDENTIFIER',

Неподдерживаемые

'ILLEGAL_TYPE'

Построение синтаксического дерева:

yacc позволяет парсить набор токенов используя методы (начинающиеся с ‘p_’) у которых в docstring’е описаны правила для этой конструкции языка.

Пример:

Данные два метода (тела методов не указаны) позволяют парсить отсутствие аргумента, один аргумент или набор из аргументов разделенных запятой)

```
def p_argDefs(p):
    """argDefs :
                | argDef COMMA argDefs
                | argDef"""
def p_argDef(p):
    """argDef : identifier
              | identifier AS typeRef"""
```

Результаты работы

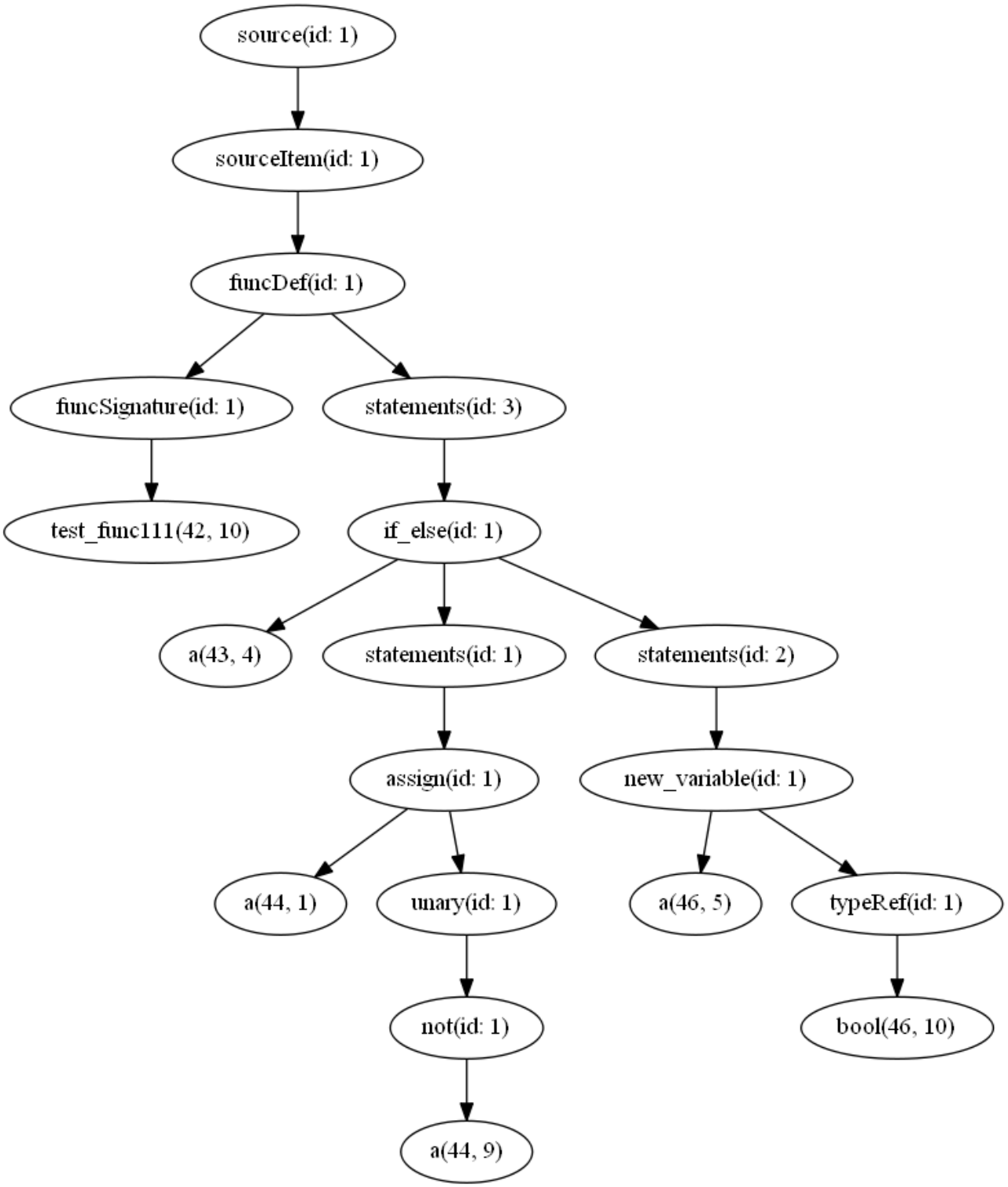
Вывод программы представлен в 3 вариантах: в виде изображения (1), текстового файла (2), более упрощенный вариант текстового файла (3). Вариант 1,3 удобны для отображения и будут представлены в отчете:

Пример работы программы:

Входные данные:

```
function test_func111 ()
if a then
a = not a;
else
dim a as bool
end if
end function
```

Изображение:



Текстовое представление:

```
source(id: 1)
└── sourceItem(id: 1)
    └── funcDef(id: 1)
        ├── funcSignature(id: 1)
        │   └── test_func111(42, 10)
        └── statements(id: 3)
            └── if_else(id: 1)
                ├── a(43, 4)
                ├── statements(id: 1)
                │   └── assign(id: 1)
                │       ├── a(44, 1)
                │       └── unary(id: 1)
                │           └── not(id: 1)
                │               └── a(44, 9)
                └── statements(id: 2)
                    └── new_variable(id: 1)
                        ├── a(46, 5)
                        └── typeRef(id: 1)
                            └── bool(46, 10)
```

Вывод

В ходе выполнения данной работы был изучен процесс построения синтаксических деревьев. Была написана программа реализующая построение по исходному файлу с текстом синтаксического дерева с узлами, соответствующими правилам варианта, задающего язык для анализа.

Были изучены полезные библиотеки для Python.