Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Отчет
кпрактическомузаданию№2
подисциплине“Системноепрограммноеобеспечение”

Выполнил студент группы P4114: Манна Рани

Преподаватель:

Кореньков Юрий

Дмитриевич

г. Санкт-Петербург

2024

Цели

Реализовать построение графа потока управления посредством анализа дерева разбора для набора входных файлов. Выполнить анализ собранной информации и сформировать набор файлов с графическим представлением для результатов анализа.

Задачи

1. *Описать структуры данных, необходимые для представления информации о наборе файлов, наборе подпрограмм и графе потока управления, где:*

    a. *Для каждой подпрограммы: имя и информация о сигнатуре, граф потока управления, имя исходного файла с текстом подпрограммы.*

    b. *Для каждого узла в графе потока управления, представляющего собой базовый блок алгоритма подпрограммы: целевые узлы для безусловного и условного перехода (по мере необходимости), дерево операций, ассоциированных с данным местом в алгоритме, представленном в исходном тексте подпрограммы*

2. *Реализовать модуль, формирующий граф потока управления на основе синтаксической структуры текста подпрограмм для входных файлов*

    a. *Программный интерфейс модуля принимает на вход коллекцию, описывающую набор анализируемых файлов, для каждого файла – имя и соответствующее дерево разбора в виде структуры данных, являющейся результатом работы модуля, созданного по заданию 1 (п.*
3. *b).*

    b. *Результатом работы модуля является структура данных, разработанная в п. 1, содержащая информацию о проанализированных подпрограммах и коллекция с информацией об ошибках*

    c. *Посредством обхода дерева разбора подпрограммы, сформировать для неё граф потока управления, порождая его узлы и формируя между ними дуги в зависимости от синтаксической конструкции, представленной данным узлом дерева разбора: выражение, ветвление, цикл, прерывание цикла, выход из подпрограммы – для всех синтаксических конструкций по варианту (п. 2.b)*

    d. *С каждым узлом графа потока управления связать дерево операций, в котором каждая операция в составе текста программы представлена как совокупность вида операции и соответствующих операндов (см задание 1, пп. 2.d-g)*

e. *При возникновении логической ошибки в синтаксической структуре при обходе дерева разбора, сохранить в коллекции информацию об ошибке и её положении в исходном тексте*

3. *Реализовать тестовую программу для демонстрации работоспособности созданного модуля*

   a. *Через аргументы командной строки программа должна принимать набор имён входных файлов, имя выходной директории*

   b. *Использовать модуль, разработанный в задании 1 для синтаксического анализа каждого входного файла и формирования набора деревьев разбора*

   c. *Использовать модуль, разработанный в п. 2 для формирования графов потока управления каждой подпрограммы, выявленной в синтаксической структуре текстов, содержащихся во входных файлах*

   d. *Для каждой обнаруженной подпрограммы вывести представление графа потока управления в отдельный файл с именем "sourceName.functionName.ext" в выходной директории, по-умолчанию размещать выходной файлы в той же директории, что соответствующий входной*

   e. *Для деревьев операций в графах потока управления всей совокупности подпрограмм сформировать граф вызовов, описывающий отношения между ними в плане обращения их друг к другу по именам и вывести его представление в дополнительный файл, по-умолчанию размещаемый рядом с файлом, содержащим подпрограмму main.*

   f. *Сообщения об ошибке должны выводиться тестовой программной (не модулем, отвечающим за анализ!) в стандартный поток вывода ошибок*

4. *Результаты тестирования представить в виде отчета, в который включить:*

   a. *В части 3 привести описание разработанных структур данных*

   b. *В части 4 описать программный интерфейс и особенности реализации разработанного модуля*

   c. *В части 5 привести примеры исходных анализируемых текстов для всех синтаксических конструкций разбираемого языка и соответствующие результаты разбора*

Описаниеработы

Работа основана на практическом задании №1. На данном этапе работы был добавлен модуль для генерации CFG, задача которого обходить AST-дерево для генерации нового дерева. Для визуализации дерева используется библиотека graphviz

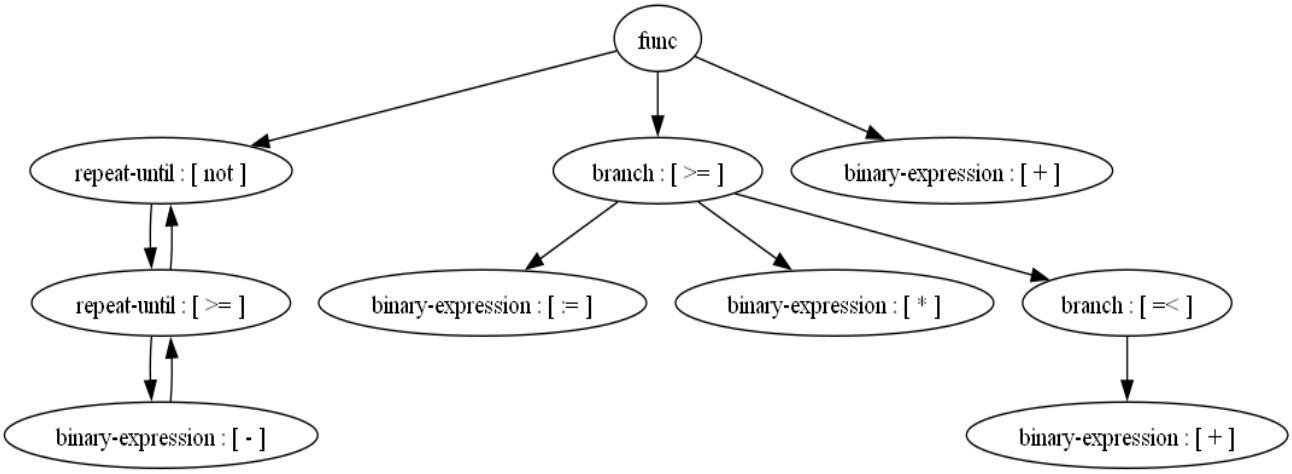
На вход подается файл с программным кодом, соответствующий грамматике, например:

```
method main(args) : int
var a : int, b : array [1, 2, 3] of int , c : str;
begin
if a >= b then c := a; else c := 0;
while (true) do
begin
repeat a := b - c;
while a >= b;
break;
end
call(expr1, expr2);
indexer [e1, e2];
end
```

В результате работы программа генерирует файл output.dot, описывающий граф:

```
digraph G { node
    [label="\N"];
    0    [label=func];
    1    [label="repeat-until : [ not ]"];
    0 -> 1;
    4        [label="branch : [ >= ]"];
    0 -> 4;
    9            [label="binary-expression : [ + ]"];
    0 -> 9;
    2            [label="repeat-until : [ >= ]"];
    1    -> 2;
    2    -> 1;
    3    [label="binary-expression : [ - ]"];
    2 -> 3;
    3 -> 2;
    5            [label="binary-expression : [ := ]"];
    4 -> 5;
    6            [label="binary-expression : [ * ]"];
    4 -> 6;
    7        [label="branch : [ =< ]"];
    4 -> 7;
    8            [label="binary-expression : [ + ]"];
    7 -> 8;
}
```

Визуализация графа в output.png:



Аспектыреализации

Пример кода генерации CFG:

```
struct cfg_node *cfg_create(struct ast_node *node, struct cfg_node
*prev_cfg) { if (node ==
    NULL) return NULL;

    node->is_visited = true; struct cfg_node *cnode = NULL; struct cfg_node
    *ret_val = NULL; switch (node->type) { case T_PROGRAM: { ret_val =
    cfg_create(node->as_program.child, cnode); break;
    } case T_EXPR_LIST: case
    T_ARGDEF_LIST:
    case T_ARRAY:  case
    T_LIT_LIST: case
    T_STMTS_LIST: {

        struct ast_node *iter = node; while (iter != NULL) { ret_val = cfg_create(iter-
        >as_list.current, prev_cfg); iter = iter->as_list.next;
        } break;
    } case T_WHILE:
    case T_REPEAT: { cnode = create_cfg_node(node);
        cnode->body = node->as_repeat.test;
```

```
    ret_val = cfg_create(node->as_repeat.body, cnode); insert_connect(cnode, ret_val-
    >connection_array);

    break;
}
case T_BRANCH: { cnode = create_cfg_node(node);
    cnode->body = node->as_branch.test;

    ret_val = cfg_create(node->as_branch.consequent, cnode); ret_val = cfg_create(node-
    >as_branch.alternate, cnode); break;
}
case T_BIN_EXPR: { cnode =
    create_cfg_node(node); cnode->body = node;
    break;
}
case T_UN_EXPR: { cnode =
    create_cfg_node(node); cnode->body = node;
    break;
}
case T_INDEXER: { break;
}
case T_CALL_EXPR: { break;
}
case T_FUNC_SIGN: { break;
}
case T_FUNC: { cnode = create_cfg_node(node);
    ret_val = cfg_create(node->as_func.body, cnode); break;
}
case T_IDENT: { break;
}
case T_BODY: { ret_val = cfg_create(node->as_body.statement, prev_cfg); break;
}

case T_STMT: { ret_val = cfg_create(node->as_statement.some_node, prev_cfg); break;
}
case T_EXPR: {
```

```
    ret_val = cfg_create(node->as_expr.some_node,  prev_cfg); break;
}
case T_TYPE_REF:  { break;
} default: {
break;
}
}
if (cnode && prev_cfg) { insert_connect(cnode,  prev_cfg-
    >connection_array);
}
return ((cnode) ? cnode : ret_val); }
```

Результаты

Результатом выполнения работы является программа, способная строить AST-дерево для кода, соответствующего грамматике по заданию, а затем строить по нему CFG-дерево.

Выводы

В ходе выполнения практической работы №2 я изучил работу с CFG-деревом и библиотекой graphviz. Все встреченные мной в процессе трудности были преодолены.