ПРИМЕНЕНИЕ ПРОГРАММНЫХ ИНСТРУМЕНТОВ С ОТКРЫТЫМ ИСХОДНЫМ КОДОМ ДЛЯ АНАЛИЗА ИСХОДНЫХ ТЕКСТОВ ПРОГРАММ

Пустыгин А.Н., Кузьминых К.М., Ковалевский А.А.,
Пустыгина Е.А., Егоров Д.Ю.

Челябинск, Челябинский государственный университет

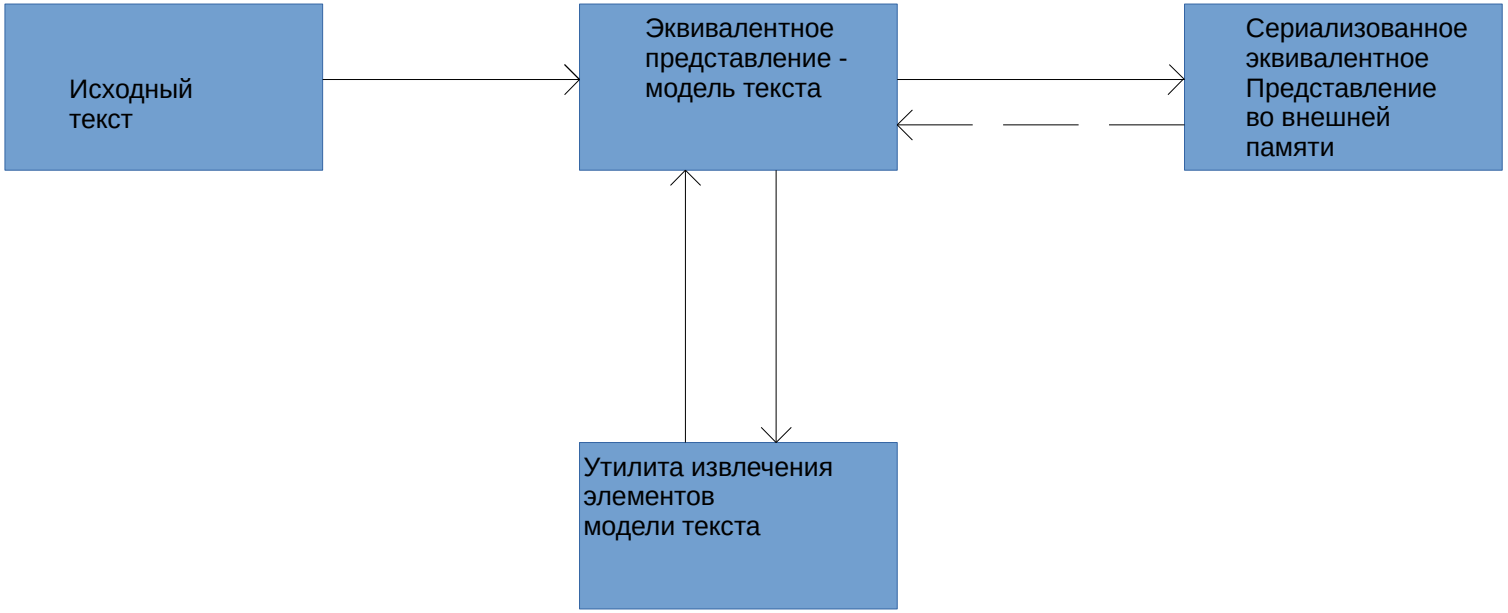
Инструменты анализа исходного текста

С целью актуализации выполняемых разработок были предприняты усилия для обзора открытых инструментов, пригодных в качестве базы сравнения с разрабатываемыми прототипами. Были рассмотрены существующие открытые инструменты для анализа, использующие построение синтаксического дерева разбора (AST):

1) PyCharm — IDE с открытой версией (подписная free licence) [1].
2) Vulture — анализатор кода в программах Python [2], использует модуль ast стандартной библиотеки и создает абстрактные синтаксические деревья для всех файлов исходного кода в проекте.
3) ObjectWeb ASM (частью входит в Java Development Kit 6) [3] — API для представления кода JAVA в виде дерева.
4) Joern -Анализатор исходного кода, способный построить: абстрактное синтаксическое дерево, граф управления, граф вызовов, граф структуры каталога исходного кода, граф программных зависимостей для исходного текста на языке C++.

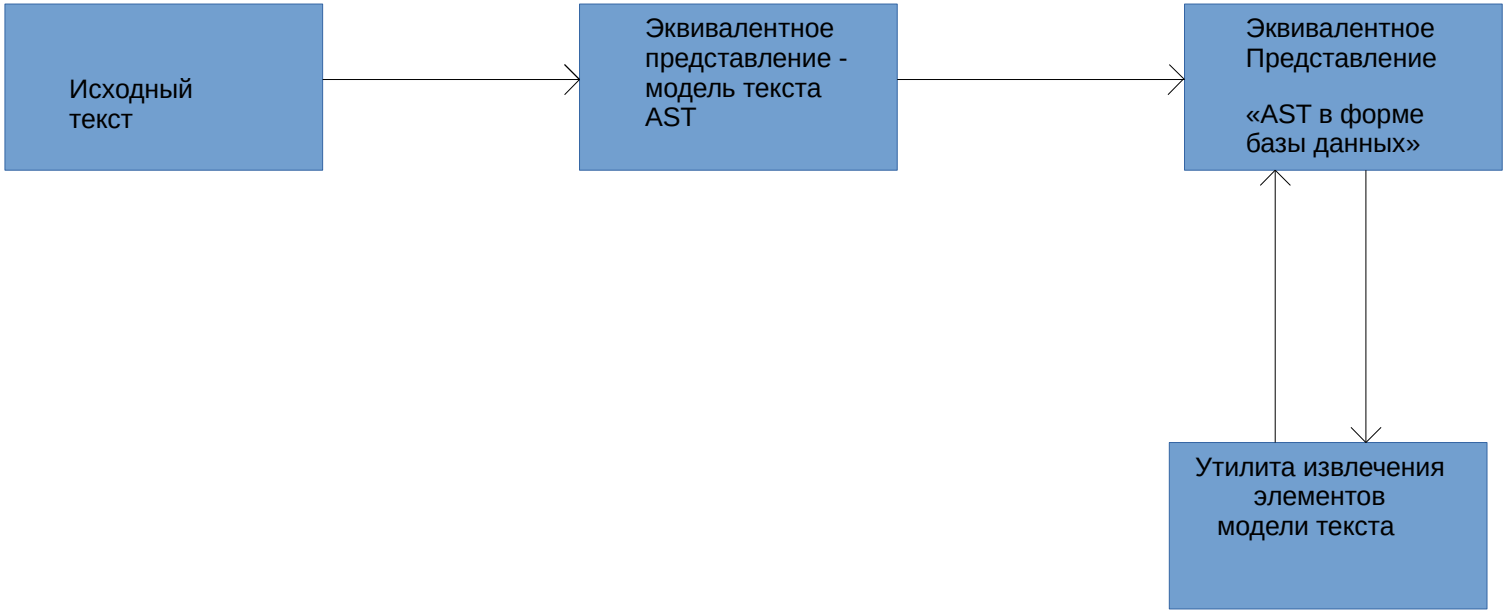
СТРУКТУРНЫЕ СХЕМЫ способов ПОЛУЧЕНИЯ ДАННЫХ ОБ ИСХОДНОМ КОДЕ

1. Схема использования Эквивалентного Представления, размещенного в оперативной памяти



СТРУКТУРНЫЕ СХЕМЫ ПОЛУЧЕНИЯ ДАННЫХ ОБ ИСХОДНОМ КОДЕ

2. Схема использования Эквивалентного Представления в форме базы данных



Проект по применению Joern, как наиболее функционального, для решения типовых анализа потока управления

Для обработки информации, полученной с помощью инструмента Joern, применялась написанная программа пост-обработки на Python. В функционал пост-обработки входят:

— построение трассы исполнения программы по исходному тексту;

— получение списка всех информационных объектов программы по исходному тексту;

— проверка наличия заданных конструкций в исходном тексте программы;

— формирование перечня маршрутов исполнения функциональных объектов по исходному тексту программы;

—контроль связей функциональных объектов в исходном тексте программы по информации;

— контроль связей функциональных объектов в исходном тексте программы по управлению.

Параметры запуска и результаты тестирования программы пост-обработки доступны в открытом источнике [6].

Проект по применению Joern, как наиболее функционального, для решения типовых анализа потока управления - 2

Под функциональными объектами понимаются объекты классов, локальные и глобальные переменные проекта, непосредственно участвующие в алгоритме. Под маршрутом исполнения функционального объекта понимается его «линия жизни» от момента создания до указанного места в исходном коде. Результатом исполнения скрипта пост-обработки будет перечень всех маршрутов исполнения, в которых, так или иначе встречается использование указанного функционального объекта, либо сообщение о том, что такие маршруты отсутствуют.

Программа пост-обработки может контролировать связи функциональных объектов проекта по управлению и по информации. Для этих задач необходимо выбрать два функциональных объекта: исследуемый и второстепенный.

Под связью функциональных объектов по управлению понимается любое воздействие исследуемого объекта на второстепенный, в том числе: модификация (чтение, запись) данных второстепенного объекта, вызов методов второстепенного объекта и любое другое воздействие на данные второстепенного объекта.

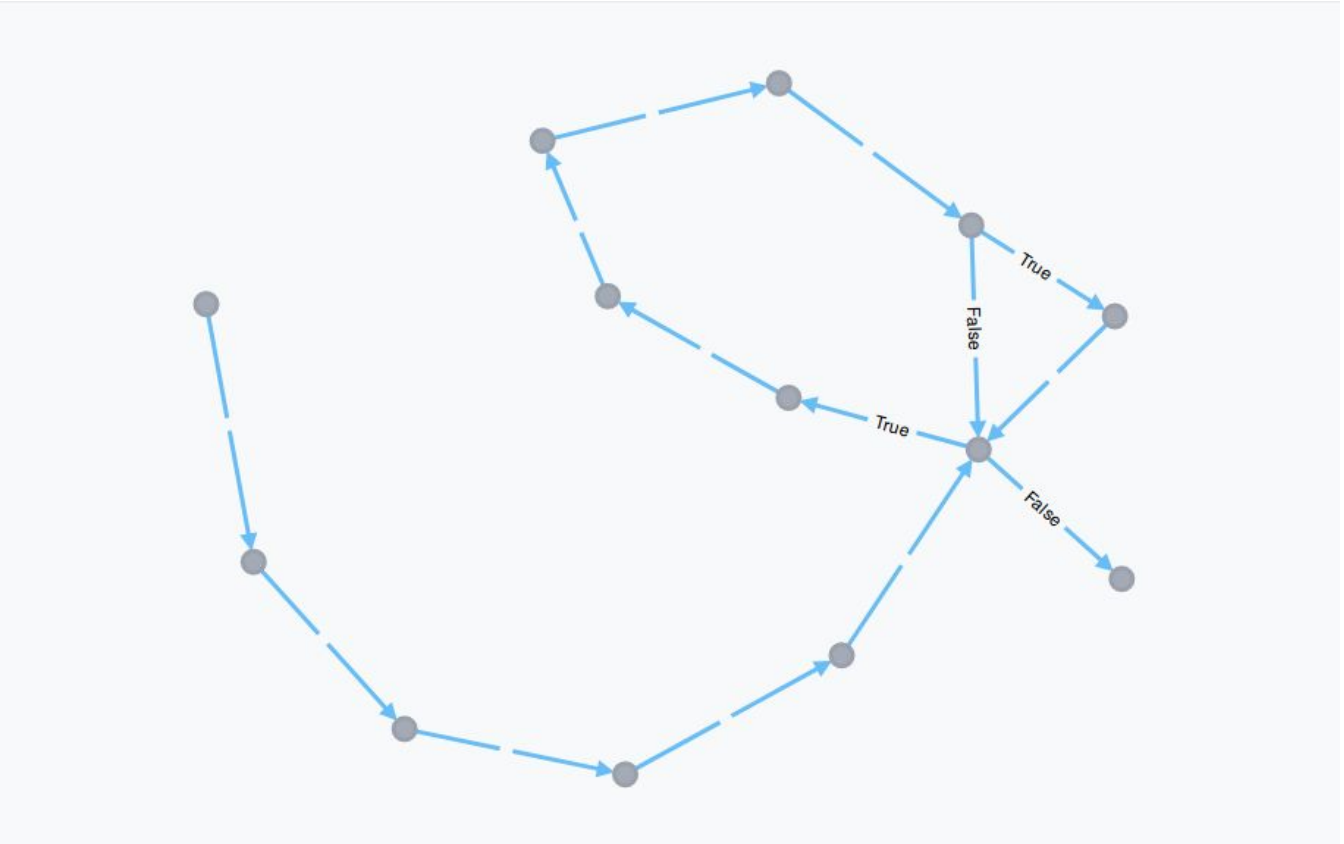
Под связью функциональных объектов по информации подразумевается любое взаимодействие второстепенного объекта на данные исследуемого объекта.

ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ ВИЗУАЛИЗАЦИИ JOERN

```
ifstream file("graphFile");

    unsigned int n = 0;
    unsigned int oriented;
    file >> n >> oriented;

    G.resize(n);
    while (!file.eof()) {
        int a, b;
        file >> a;
        file >> b;
        G[a].push_back(b);
        if(oriented == 0) {
            G[b].push_back(a);
        }
    }
    file.close();
```



Получение графа исполнения участка исходного кода с помощью графовой базы данных Neo4j и инструмента Joern.

1. Построение графовой базы данных с помощью инструмента Joern.
Команда: joern -outdir <out-dir> <src-dir>
2. Открытие базы данных с помощью СУБД Neo4j и выполнение запроса В Neo4j-shell выполнить запрос для получения трассы в текстовом виде:
MATCH (n1 {code:"CODE_1", isCFGNode:"True"}) MATCH (n2 {code:" CODE_2", isCFGNode:"True"}) MATCH path = ((n1)-[:FLOWS_TO..]-(n2)) RETURN extract(n IN nodes(path) | n.code) AS codes, extract(n IN relationships(path) | n.flowLabel) AS flowLabels;*
где CODE_1 – начальная инструкция в трассе, CODE_2 – конечная инструкция в трассе.

Тестирование базы данных исходного кода Neo4j посредством инструмента постобработки

построить трассу исполнения участка кода, либо сообщить, что такой трассы не существует

```cpp
ifstream file("graphFile");   // точка начала поиска трассы

unsigned int n = 0;
unsigned int oriented;
file >> n >> oriented;

G.resize(n);

while (!file.eof()) {
    int a, b;
    file >> a;
    file >> b;

    G[a].push_back(b);
    if(oriented == 0) {
        G[b].push_back(a);
                        }
                    }

file.close();       // точка завершения поиска трассы
```

Параметры запуска:

"code_1" : "ifstream file ("graphFile") ;"
"code_2" : "file . close ( )"

Результаты тестирования:

==========

Trace 1

121:1 ifstream file ("graphFile") ; -- CompoundStatement -->
123:1 unsigned int n = 0 ; -- CompoundStatement -->
124:1 unsigned int oriented ; -- CompoundStatement -->
125:1 file >> n >> oriented -- CompoundStatement -->
127:1 G . resize (n) -- CompoundStatement -->
129:8 ! file . eof ( ) -- WhileStatement:False -->
140:1 file . close ( )

==========

Trace 2

121:1 ifstream file ("graphFile") ; -- CompoundStatement -->
123:1 unsigned int n = 0 ; -- CompoundStatement -->
124:1 unsigned int oriented ; -- CompoundStatement -->
125:1 file >> n >> oriented -- CompoundStatement -->
127:1 G . resize (n) -- CompoundStatement -->
129:8 ! file . eof ( ) -- WhileStatement:True -->
130:2 int a , b ; -- CompoundStatement -->
131:2 file >> a -- CompoundStatement -->
132:2 file >> b -- CompoundStatement -->
134:2 G [a] . push_back (b) -- CompoundStatement -->
135:5 oriented == 0 -- IfStatement:False -->
129:8 ! file . eof ( ) -- WhileStatement:False -->
140:1 file . close ( )

==========

Trace 3

121:1 ifstream file ("graphFile") ; -- CompoundStatement -->
123:1 unsigned int n = 0 ; -- CompoundStatement -->
124:1 unsigned int oriented ; -- CompoundStatement -->
125:1 file >> n >> oriented -- CompoundStatement -->
127:1 G . resize (n) -- CompoundStatement -->
129:8 ! file . eof ( ) -- WhileStatement:True -->
130:2 int a , b ; -- CompoundStatement -->
131:2 file >> a -- CompoundStatement -->
132:2 file >> b -- CompoundStatement -->
134:2 G [a] . push_back (b) -- CompoundStatement -->
135:5 oriented == 0 -- IfStatement:True -->
136:3 G [b] . push_back (a) -- CompoundStatement -->
129:8 ! file . eof ( ) -- WhileStatement:False -->
140:1 file . close ( )

==========

Elapsed time = 86.5544698447 seconds

Машина: двухядерный Intel® Core™ i7 processor 2,8 ГГц , ОЗУ 16 Гб

Тестирование базы данных исходного кода Neo4j

посредством инструмента постобработки

получить список всех функциональных объектов проекта с указанием идентификатора объекта, места его создания, исходного кода создания и наименования самого объекта (идентификатор).

Последовательность действий оператора:

1. Команда для запуска python main.py sfo functional_objects.txt.

Elapsed time = 396.105171529 seconds

Тестирование базы данных исходного кода Neo4j посредством инструмента постобработки

сформировать перечень маршрутов исполнения функциональных объектов.

Последовательность действий оператора:

1. Получить список функциональных объектов проекта, команда для запуска python main.py sfo functional_objects.txt.
2. В конфигурационном файле заполнить поля “code_1” и “code_2” в секции “code_trace”. В поле “code_1” необходимо указать исследуемый объект.
3. Запустить программу командой python main.py ct

```cpp
ifstream file("graphFile");

unsigned int test = 123;
unsigned int n = 0;
unsigned int oriented;
file >> n >> oriented;

G.resize(n);

while (!file.eof()) {
    int a, b;
    file >> a;
    file >> b;

    G[a].push_back(b);
    if(oriented == 0) {
        G[b].push_back(a);
    }
}

file.close();
```

Параметры запуска:

"code_1" : "ifstream file ("graphFile") ;"
"code_2" : "file . close ( )"
"functional_object_trace" : 1

Результат тестирования:

get_trace_main start
CODE_1 = ifstream file (\"graphFile\") ;
CODE_2 = file . close ( )
Get all path... done.
Get types of statements... done.
Get one functional object... done.
Get symbol of functional object... done.
Start checking fucntional object traces...
Number of object for check: 1
Symbols for check: file
done.

time = 100.567015306 seconds

Elapsed time = 100.567015306 seconds

Тестирование базы данных исходного кода Neo4j

посредством инструмента постобработки

найти все участки взаимодействия двух функциональных объектов, связанных по управлению

Последовательность действий оператора:

1. В конфигурационном файле заполнить поля "main_object" и "secondary_object" в секции "functional_management_control" двумя объектами, полученными при помощи вызова программы с параметром sfo.
2. Запустить программу командой python main.py fmc

```cpp
void testFunc() {
	string filename;
	string strToWrite("test");
	filename = "data.txt";

	ofstream file(filename);

	int someNumber = 0;
	for (int i = 0; i++; i < 100) {
		someNumber = i + 1;
		file.write(strToWrite.c_str(), strToWrite.length());
		file << strToWrite;
		strToWrite += filename;
	}

	file.close();

	return;
}
```

Examples of interaction between objects

For trace 0

162:1 ofstream file (filename) ;

For trace 1

162:1 ofstream file (filename) ;

Elapsed time = 116.770251711 seconds

Тестирование базы данных исходного кода Neo4j

посредством инструмента постобработки

найти все участки взаимодействия двух функциональных объектов, связанных по информации.

Последовательность действий оператора:

1. В конфигурационном файле заполнить поля "main_object" и "secondary_object" в секции "functional_information_control" двумя объектами, полученными при помощи вызова программы с параметром sfo.
2. Запустить программу командой python main.py fmc

```cpp
void testFunc() {
	string filename;
	string strToWrite("test");
	filename = "data.txt";

	ofstream file(filename);

	int someNumber = 0;
	for (int i = 0; i++; i < 100) {
		someNumber = i + 1;
		file.write(strToWrite.c_str(), strToWrite.length());
		file << strToWrite;
		strToWrite += filename;
	}

	file.close();

	return;
}
```

Параметры:

"main_object" : "ofstream file (filename) ;"
"secondary_object" : "string strToWrite (\\\"test\\\") ;"

Examples of interaction between objects
For trace 0
No
For trace 1
167:2 file . write (strToWrite . c_str ( ) , strToWrite . length ( ))
168:2 file << strToWrite
Elapsed time = 106.563243658 seconds

Полученные результаты, выводы и заключение

1. В результате опытного применения «наилучшего» прототипа, основанного на AST, представленной в виде БД, выяснилось, что производительность этого инструмента весьма низка и не может конкурировать с инструментами, использующих Эквивалентное Представление в памяти.

2. Глубина построения AST, согласно имеющемуся описанию, не позволяет связывать графы передачи управления отдельных модулей.

3. Изобразительные возможности визуализации, заявленные разработчиками, не отвечают рядового пользователя.