Структуризация потока управления путём функционализации в задачах декомпиляции

Забранский Дмитрий Юрьевич, 461 гр.

Научный руководитель: к.ф.-м.н. Д.Ю. Булычев
Рецензент: к.ф.-м.н. О.А. Плисс

Введение

- Проект «Декомпиляция JVM байт-кода»
- Актуальность
  - Поддержка программного обеспечения
  - Безопасность

Постановка задачи

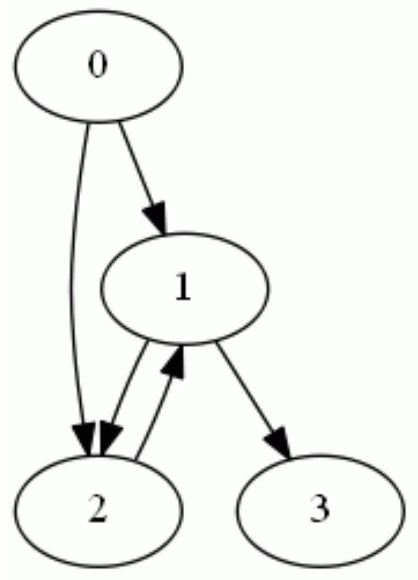
В ходе работы были сформулированы следующие задачи:

- Изучение спецификации JVM и библиотеки ASM;
- Разбор байт-кода;
- Построение и визуализация графа потока управления;
- Реализация функционализации потока управления;
- Исследование результатов функционализации.

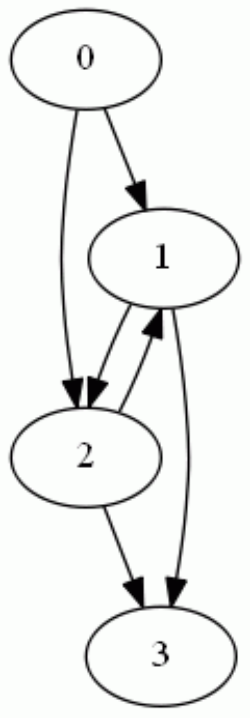
Анализ потока управления

- Граф потока управления
- Восстановление высокоуровневых структурных конструкций
- Структуризация
- Неприводимая область и несводимость графа

Несводимые графы



Пример с GOTO



Классический пример

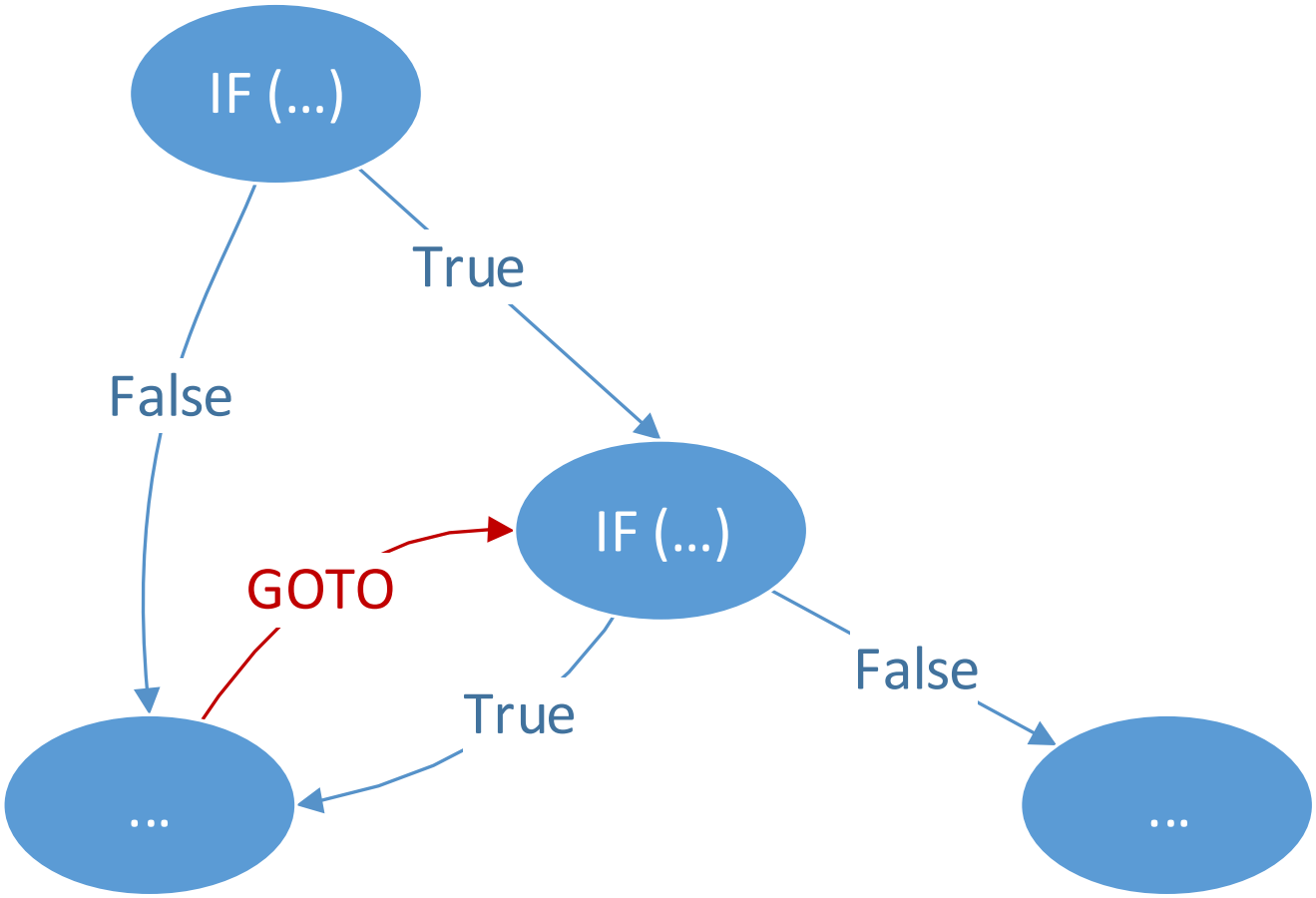
Способы структуризации

- Преобразование графа к сводимому
- Диспетчер переходов
- Функционализация

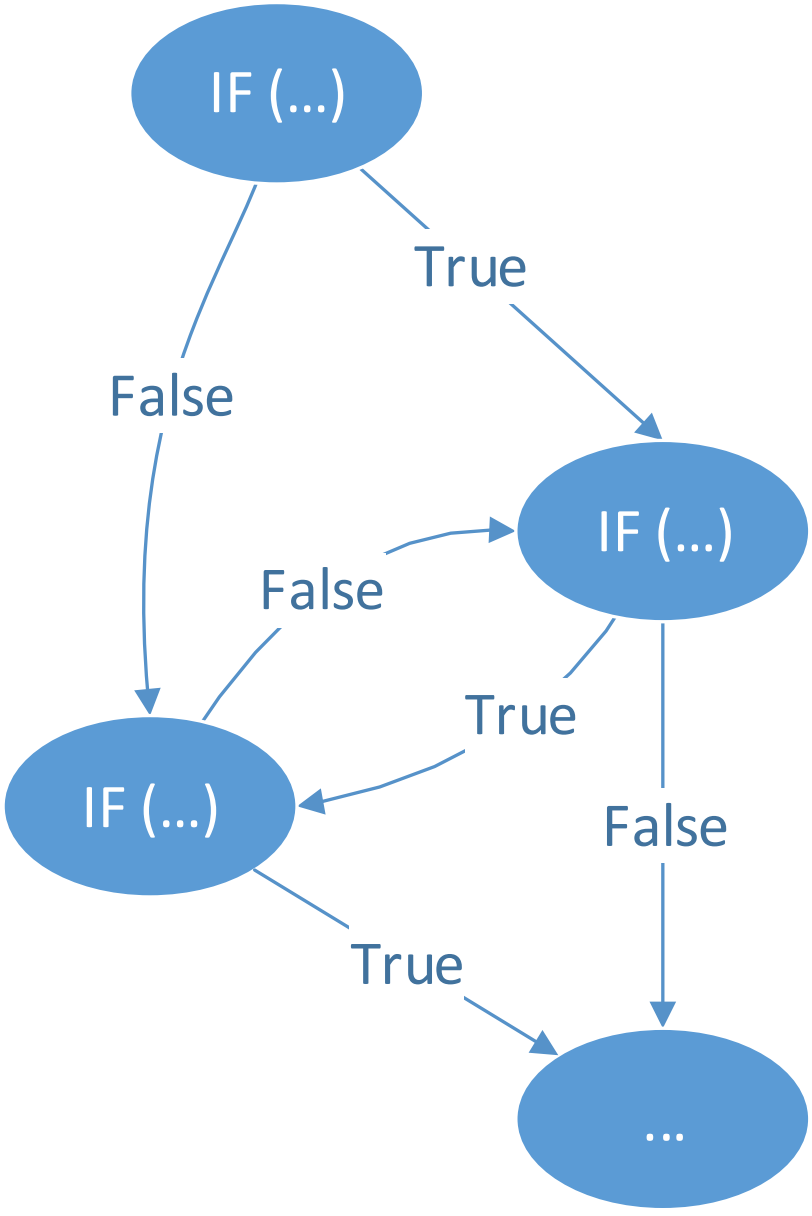
Этапы работы

- Синтаксический анализ структурных конструкций
- Построение и визуализация графа потока управления
- Реализация функционализации потока управления

Пример с GOTO



Классический пример



Функционализация

```
1  package tests;
2  public class Test2 {
3      public int main() {
4          return new Object() {
5              int y1 = 0;
6              private int start() {
7                  y1 = y1 + 1;
8                  if (...) {
9                      return fnode_1();
10                 } else {
11                     return fnode_2();
12                 }
13             }
14             private int fnode_1() {
15                 if (...) {
16                     return fnode_2();
17                 } else {
18                     return fnode_3();
19                 }
20             }
21             private int fnode_2() {
22                 y1 = y1 + 1;
23                 return fnode_1();
24             }
25             private int fnode_3() {
26                 return y1;
27             }
28         }.start();
29     }
30 }
31
32
33
34
35
```

```
1  package tests;
2  public class Test {
3      public int main() {
4          return new Object() {
5              int y1 = 0;
6              private int start() {
7                  y1 = y1 + 1;
8                  if (...) {
9                      return fnode_1();
10                 } else {
11                     return fnode_2();
12                 }
13             }
14             private int fnode_1() {
15
16                 if (...) {
17                     return fnode_2();
18                 } else {
19                     return fnode_3();
20                 }
21             }
22             private int fnode_2() {
23
24                 if (...) {
25                     return fnode_3();
26                 } else {
27                     return fnode_1();
28                 }
29             }
30             private int fnode_3() {
31                 return y1;
32             }
33         }.start();
34     }
35 }
```

Результат работы декомпиляторов

- JAD (JAva Decompiler)

```
Parsing Test.class... Generating Test.jad
Couldn't fully decompile method main
```

- Java decompiler (Yet another fast Java decompiler)
  Вывод ошибочного кода, игнорирование переходов
- DJ Java Decompiler
  Вывод промежуточного представления JAD

Результаты

1. Реализован разбор переходов, меток и оператора множественного ветвления;
2. Реализовано построение графа потока управления;
3. Реализована визуализация графа потока управления;
4. Реализована функционализация потока управления;
5. Проведено исследование результатов функционализации;
6. Проведено сравнение работы декомпиляторов на примерах с несводимым графом потока управления.