Статический анализ фрагментированных программ на базе графа зависимостей по данным

Студент 4 курса ФИТ НГУ
Царев Василий Дмитриевич

Научный руководитель:
Власенко Андрей Юрьевич

Новосибирск, 2024

DeGSA

Dependency Graph Static Analyzer. Строит граф зависимостей по данным из AST входной LuNA-программы и запускает несколько функций-чекеров для сбора информации об ошибках в исходном коде.

Состояние на момент начала школы:

- поддержка исключительно cf и for;
- реализован поиск ошибок трёх типов;
- отсутствовала поддержка индексированных ФД;
- ограниченное связывание вершин.

Задачи и результаты

1. Написать статью для конференции ПаВТ 2024 – выполнено.

1. Доработать структуру графа, а именно:

- переработать хранение ФД в графе – выполнено;
- дополнить информацией вершины типа for – выполнено частично;
- добавить вершины типа while – не выполнено.

Переработка хранения ФД

Состояние системы до начала школы: все ФД участвуют в системе исключительно как объекты типа string. Недостатки (не включая указанные ранее):

- нет ссылок между именами (для связи ФД между структурированными ФК);

- отсутствует защита от коллизии имён.

Переработка хранения ФД

Результаты работы на школе:

- Иерархическая структура для хранения информации об идентификаторах:

```
1   sub main(){
2           df a;
3           for i = 1..2{
4                   foo(a[1]);
5                   print(a[1][1]);
6           }
7   }
8
9   sub foo(df f){
10          init(1, f);
11          print(f);
12  }
```

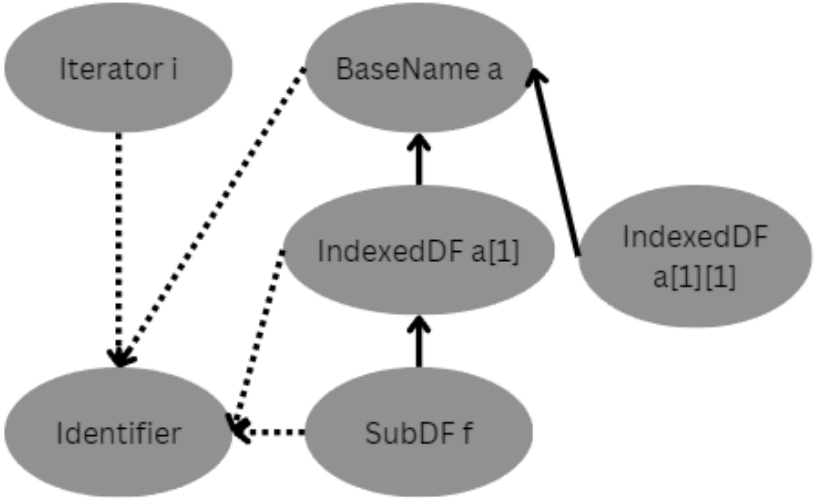

Рис. 1. Пример структуры для хранения информации о ФД

Переработка хранения ФД

- Таблица базовых имён: для каждого имени указано, где соответствующие индексированные ФД были инициализированы и использованы (с учётом количества индексов).

```
Name: c
Size: 0
Uses:
Defs: 0x564162bf65e0 0x564162bf7110
Size: 1
Uses: 0x564162bf7fb0
Defs:
Size: 2
Uses: 0x564162bf83c0
Defs:
```

Рис. 2. Пример таблицы для базового имени “c”

Переработка хранения ФД

- Таблица используется для связывания вершин вне зависимости от областей видимости и с учётом количества индексов у имени. Результирующий граф более точно отражает структуру программы и позволяет искать ошибки для большего множества программ.

```
1   sub main(){
2           df a;
3           for i = 1..2{
4                   foo(a[1]);
5                   print(a[1]);
6           }
7   }
8
9   sub foo(df f){
10          init(1, f);
11          print(f);
12  }
```

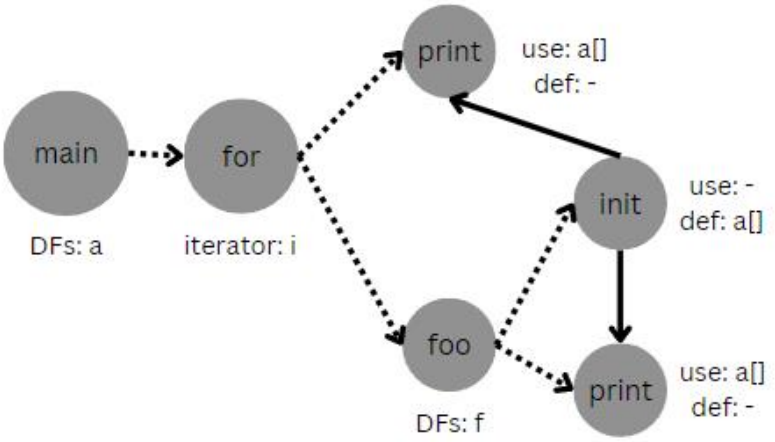

Рис. 3. Пример новой структуры графа

Доработка вершин типа for

Результаты работы на школе:

- Вершины содержат ссылки на выражения, указанные при описании ФК в программе.

- Для итераторов также создаются соответствующие объекты-идентификаторы.

Невыполненные планы:

- Отсутствует обработка логики оператора (т.е. порождение им нескольких ФВ).

Перспективы созданной модели

1. Парсинг выражений внутри индексов.

1. Поиск таких ошибок как попытка инициализации неподходящего выражения и циклическая зависимость по данным.

Планы на семестр

1. Добавить в граф типы вершин для остальных операторов.
2. Добавить вывод стека вызовов.
3. Поиск указанных ранее ошибок: циклическая зависимость по данным и попытка инициализации неподходящего выражения.
4. Внедрение DeGSA в общую систему автоматизированной отладки.

Спасибо за внимание!