Санкт-Петербургский государственный университет
Кафедра системного программирования

Автоматический синтез объектов для символьного исполнения

Максим Алексеевич Паршин

Научный руководитель: к.ф.-м.н. Д.А. Мордвинов, доцент кафедры системного программирования

Санкт-Петербург
2023

Введение

```
1  void Foo(int x, int y, int z)
2  {
3          int a = 0;
4
5          if (x < 42)
6          {
7                  a = y + z;
8          }
9
10         if (a > 73)
11         {
12                 throw new Exception();
13         }
14 }
```

- Символьное исполнение — исполнение кода не на конкретных значениях входных данных, а на символьных переменных
- Для каждого пути исполнения — условие на символьные переменные, при выполнении которого он достигается
- Например, функция `Foo` выбрасывает исключение в 12 строке, если $\pi = x < 42 \ \wedge \ y + z > 73$
- Используется для автоматической генерации тестов и поиска ошибок
- V# — символьная машина для .NET[a]

[a] https://github.com/VSharp-team/VSharp

Введение

```
1  public class StaticsForType<T> where T:class
2  {
3      private readonly List<T> myList = new
           List<T>();
4      private event Action? Changed;
5      ...
6      public void ForEachValue(Action action)
7      {
8        lock (myList)
9        {
10         Changed += action;
11       }
12       action();
13     }
14     ...
15 }
```

- Другой пример, из библиотеки JetBrains.Lifetimes[a]
- V# находит ошибку: при *myList* == *null* метод `ForEachValue` выбрасывает `ArgumentNullException` в строке 8
- Такой объект `StaticsForType` успешно создаётся рефлексией
- Однако *myList* не может быть равен *null* при реальном использовании данного класса
- Необходимо создавать объекты с заданными свойствами так же, как это делал бы пользователь

[a] https://github.com/JetBrains/rd

Постановка задачи

Целью данной работы является реализация автоматического синтеза объектов на основе механизма символьного исполнения в символьной виртуальной машине V#

Задачи (практика):

- Провести обзор методов синтеза объектов, используемых в различных инструментах генерации тестов
- Разработать алгоритм синтеза объектов, основанный на прямом символьном исполнении
- Реализовать разработанный алгоритм в символьной виртуальной машине V#
- Провести эксперименты для определения эффективности реализованного алгоритма

Задачи (BKP):

- Разработать алгоритм синтеза объектов, основанный на двунаправленном символьном исполнении
- Реализовать разработанный алгоритм в символьной виртуальной машине V#
- Провести эксперименты, сравнить эффективность реализованных алгоритмов

Основные понятия

- *Последовательность методов* — последовательность троек $(M_i, Ret_i, Args_i)$
  - M_i — метод
  - Ret_i — множество $\{res_{i_1}, ..., res_{i_{r_i}}\}$ переменных непримитивных типов, в которые M_i возвращает значения
  - $Args_i$ — множество $\{arg_{i_1}, ..., arg_{i_{a_i}}\}$ аргументов параметров метода. $Args_i \subset Ret_1 \cup ... \cup Ret_{i-1} \cup Prim \cup \{null\}$, где $Prim$ — множество значений примитивных типов
- *«Скелет» последовательности* — аргументы могут принимать значение *hole*
- Задача: для метода $M_{target}(p_1, ..., p_n)$ и состояния его символьного исполнения s с условием пути $\pi(p_1, ..., p_n)$ найти последовательность методов $\{(M_i, Ret_i, Args_i)\}_{1...l}$ такую, что существуют значения $\{arg_1, ..., arg_n\} \subset Ret_1 \cup ... \cup Ret_l \cup Prim \cup \{null\}$, и после исполнения последовательности выполняется $\pi(arg_1, ..., arg_n)$

Существующие подходы к синтезу объектов

- *Automated Testing of Classes, 2000*
  - Генерация тестов, реализующих определённые def-use пары
  - Построение последовательности вызовов методов в обратном порядке
  - Методы последовательности исполняются символьно
  - Отсутствуют эксперименты, используются устаревшие инструменты
- *Symstra: A Framework for Generating Object-Oriented Unit Tests using Symbolic Execution, 2004*
  - Перебираются всевозможные последовательности методов и исполняются символьно с переменными примитивных типов
  - Состояния с изоморфными графами объектов объединяются
  - Отсутствует возможность создания вложенных объектов
- *Synthesizing Method Sequences for High-Coverage Testing, 2011*
  - Основа — символьная машина *Pex* для .NET
  - Последовательность методов генерируется для определённого пути исполнения
  - Комбинация статического анализа и динамического символьного исполнения
  - Тестирование на крупных проектах

Существующие подходы к синтезу объектов

- *Efficient Synthesis of Method Call Sequences for Test Generation and Bounded Verification, 2022*
  - Подход, схожий с *Symstra*: построение полного графа переходов между различными состояниями объектов
  - В процессе строится последовательность вызовов, методы исполняются символьно
  - Одно состояние — один тест
- Подходы, основанные на генетических алгоритмах
  - *Improving Structural Testing of Object-Oriented Programs via Integrating Evolutionary Testing and Symbolic Execution, 2008*
  - *Graph-Based Seed Object Synthesis for Search-Based Unit Testing, 2021*

Описание алгоритма

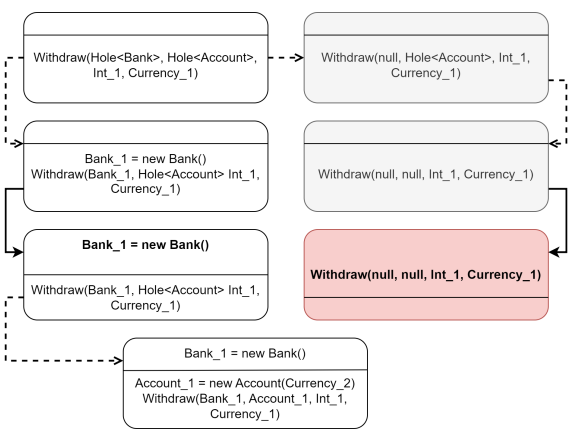
- Состояние алгоритма — это тройка $(s_{seq}, Current, Upcoming)$
  - $Current$ — построенная на данный момент последовательность методов
  - $Upcoming$ — «скелет» последовательности методов, которая в будущем будет сконкатенирована с $Current$
  - s_{seq} — состояние символьного исполнителя после символьного исполнения последовательности $Current$
- Возможны следующие переходы между состояниями
  - Если s_{seq} в текущий момент находится в одном из методов, сделать шаг символьного исполнения внутри метода
  - Если на вершине $Upcoming$ лежит элемент $(M_i, Ret_i, Args_i)$, и $hole \notin Args_i$, снять данный элемент со стека, добавить его к $Current$, сделать шаг символьной машины внутрь M_i
  - Если на вершине $Upcoming$ лежит элемент $(M_i, Ret_i, Args_i)$, то некоторым образом выбрать новый элемент «скелета» $(M_j, Ret_j, Args_j)$ и положить его на стек $Upcoming$

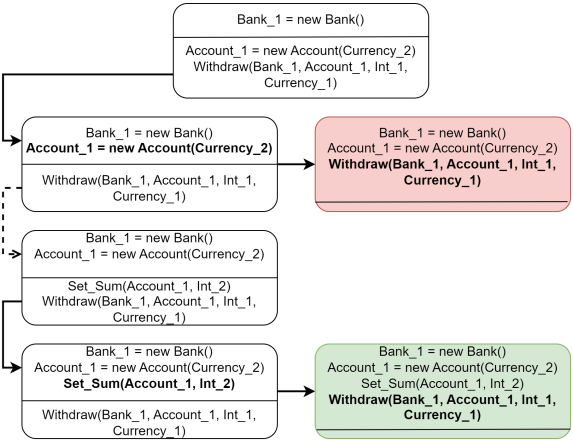
Пример работы алгоритма

```
1  public class Bank
2  {
3      public void Withdraw(Account account,
           int sum, Currency currency)
4      {
5          if (sum <= 0 || account.Currency !=
               currency)
6          {
7              throw new Exception();
8          }
9
10         if (account.Sum < sum) return;
11
12         account.Sum -= sum;
13     }
14 }
```

```
1  public enum Currency
2  {
3      RUB,
4      USD
5  }
6
7  public class Account
8  {
9      public int Sum { get; set; }
10
11     public Currency Currency { get; }
12
13     public Account(Currency currency)
14     {
15         Currency = currency;
16     }
17 }
```

Пример работы алгоритма (продолжение)





Особенности реализации

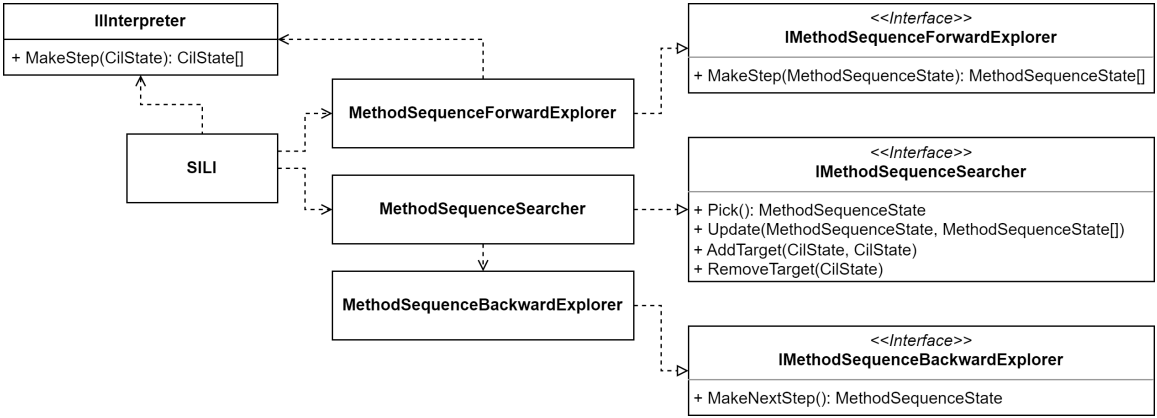


Рис.: Диаграмма реализованных классов

Эксперименты

- На данный момент пространство поиска ограничено конструкторами и методами `Set` свойств
- Синтетические тесты: методы, принимающие на вход объекты с вложенной структурой

Количество методов	20
Общее количество сгенерированных тестов	71
Количество тестов, для которых последовательность может быть сгенерирована	70
Количество тестов, для которых последовательность была сгенерирована	70
Максимальная длина последовательности	3

Таблица: Результаты экспериментов

Результаты

В ходе данной работы были получены следующие результаты

- Проведён обзор методов синтеза объектов, используемых в различных инструментах генерации тестов
- Разработан алгоритм синтеза объектов, основанный на прямом символьном исполнении
- Разработанный алгоритм реализован в символьной виртуальной машине V#
- Проведены эксперименты для определения эффективности реализованного алгоритма

В рамках ВКР планируется

- Разработать алгоритм синтеза объектов на основе механизма двунаправленного символьного исполнения
- Реализовать разработанный алгоритм в символьной виртуальной машине V#
- Провести эксперименты, сравнить эффективность реализованных алгоритмов

Пример символьного исполнения

```
1  void Foo(int x, int y, int z)
2  {
3          int a = 0;
4
5          if (x < 42)
6          {
7                  a = y + z;
8          }
9
10         if (a > 73)
11         {
12                 throw new Exception();
13         }
14 }
```