Санкт-Петербургский государственный университет
Кафедра системного программирования

Символьная виртуальная машина .NET на основе платформы USVM

Пожарский Роман Николаевич

Санкт-Петербургский государственный университет

9 июня 2025 г.

Научный руководитель: к. ф-м. н. Д.А. Мордвинов, доцент кафедры системного программирования
Консультант: М.П. Костицын, ведущий разработчик, ООО «ИИнтелКод»
Рецензент: В. Соболь, старший академический консультант, ООО «КоулмэнТех»

Санкт-Петербург
2025

Введение

Символьное исполнение

Техника анализа кода, позволяющая исполнить функцию в условиях неопределённости входных данных.

Условия, зависящие от входных данных, кодируются в логику 1-го порядка и проверяются SMT-решателями.

```
public int Abs(int x) {
    var r = x;
    if (x < 0) {
        r = -x;
     }
    return r;
}
```

Области применения:

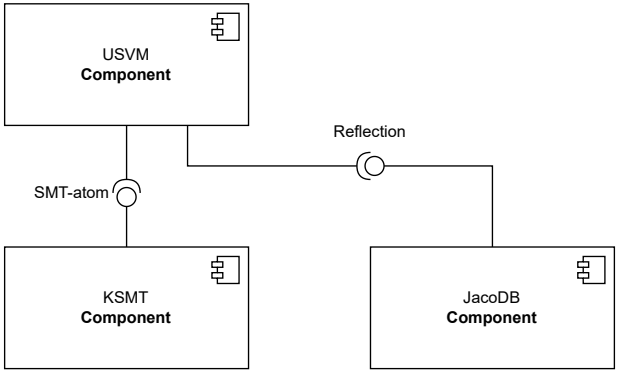
1. верификация
2. **автоматическая генерация тестов**

Введение

Платформа USVM

Символьный анализ кода на Java, Kotlin, Python, Go

1. JacoDB — библиотека для получения информации о коде
2. KSMT — библиотека для взаимодействия с решателями
3. USVM — символьное исполнение



Постановка задачи

Цель: создание системы автоматической генерации тестов на основе символьного исполнения для платформы .NET при помощи платформы USVM

Задачи:

1. провести обзор инструментов для генерации тестов и способов моделирования символьной памяти с небезопасным контекстом
2. спроектировать архитектуру символьной виртуальной машины USVM-NET
3. реализовать символьную виртуальную машину USVM-NET
4. провести интеграционное тестирование символьной машины

Инструменты автоматической генерации тестов

1. V#
   - динамическое символьное исполнение
   - конкретное исполнение для ускорения анализа
   - генерация Unit-тестов на языке C# по финальным состояниям символьной машины
2. IntelliTest
   - динамическое символьное исполнение
   - конкретное исполнение для «направления» символьного исполнения
   - генерация тестов на языке C# по финальным состояним символьной машины
   - только для Visual Studio Enterprise

Символьная память

Модель памяти формализует понятие ссылки (указателя) и операции с памятью.

Для полноты необходим анализ псевдонимов.

```
public int Foo(int[] a, int[] b) {
    Debug.Assert(b[0] != 1)
    a[0] = 1;
    if (b[0] == 1) return 1;
    return 0;
}
public void Bar(object[] a) { ... }

var a = new string[] {"asd"};
Bar(a);
```

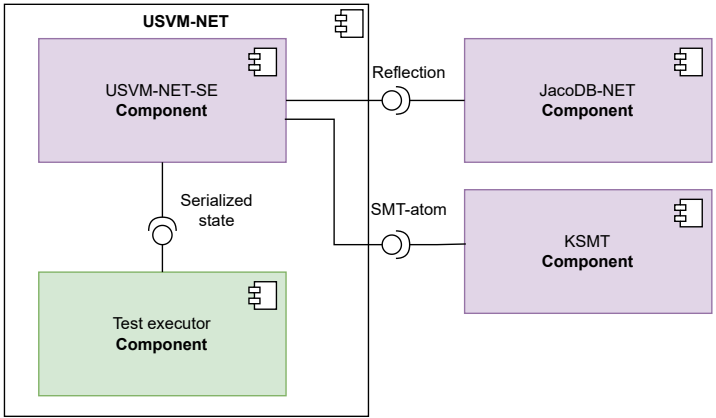
Способы моделирования символьной памяти

Символьная память V# и USVM

- V# (.NET)
  - память представлена коллекцией **регионов**
  - регионы — непересекающиеся области памяти
  - анализ указателей на основе ключей памяти (a[i], a.Field)
  - конкретная память
  - поддерживает небезопасный контекст
- USVM
  - более продвинутый алгоритм анализа псевдонимов
  - нет конкретной памяти
  - произвольный доступ к регионам

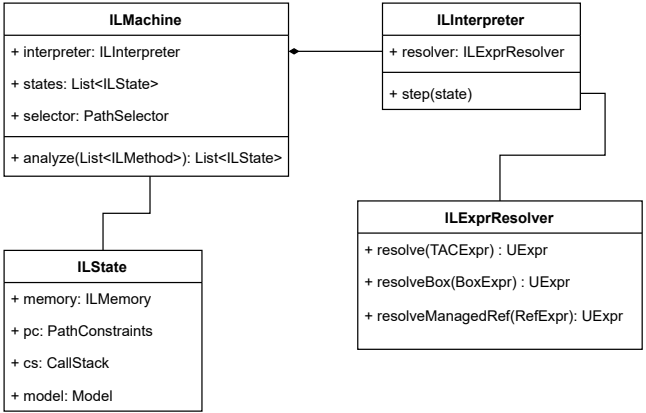
Архитектура проекта USVM-NET

- JacoDB-NET реализован вне рамок работы
- в рамках работы были добавлены компоненты USVM-NET и Test Executor



Символьная машина

- память состояния UMemory была расширена ILMemory
  - дополнительные регионы для статических полей, примитивов на куче после box
  - стек регистров (переменных), поддерживающий обращение к разным фреймам
  - поддержка небезопасного контекста на основе подхода V#: операции по небезопасным ключам сводятся к операциям по безопасным



Система генерации тестов

1. символьная машина после анализа возвращает список финальных состояний
2. при помощи модели и памяти воссоздаются параметры исследуемой функции в виде сущностей DSL
3. для валидации тесты воспроизводятся в **Test executor**, сравниваются результаты символьного и конкретного исполнения, считается покрытие

Интеграционное тестирование

Проект V# насчитывает свыше 1000 интеграционных тестов на разные категории, часть была взята из этого проекта.
Не поддерживаются символьные дженериковые объекты, делегаты, многомерные массивы, Enum-ы, циклы с символьной границей

Категория	Кол-во методов	Время анализа, с		Сгенерировано тестов		Воспроизведенные, %		Покрытие, %	
		USVM-NET	V#	USVM-NET	V#	USVM-NET	V#	USVM-NET	V#
Arithmetic	106	5.16 $\pm$0.15	2.50 $\pm$0.18	147	130	98	100	81	91
Logic	75	2.37 $\pm$0.05	0.52 $\pm$0.01	75	75	100	100	98	98
Arrays	36	3.84 $\pm$0.15	3.76 $\pm$0.11	163	112	96	100	70	96
Unsafe	28	3.24 $\pm$0.08	2.80 $\pm$0.14	49	50	96	100	82	93
Boxing	18	2.61 $\pm$0.22	0.1 $\pm$0.05	19	20	84	100	80	94
Exceptions	18	2.83 $\pm$0.15	0.81 $\pm$0.01	34	36	100	100	68	80
Structs	9	2.56 $\pm$0.08	0.69 $\pm$0.03	15	12	100	100	90	90
Classes	8	2.34 $\pm$0.06	0.69 $\pm$0.01	15	11	100	100	83	83
Calls	5	2.09 $\pm$0.12	0.67 $\pm$0.02	12	6	100	100	85	74

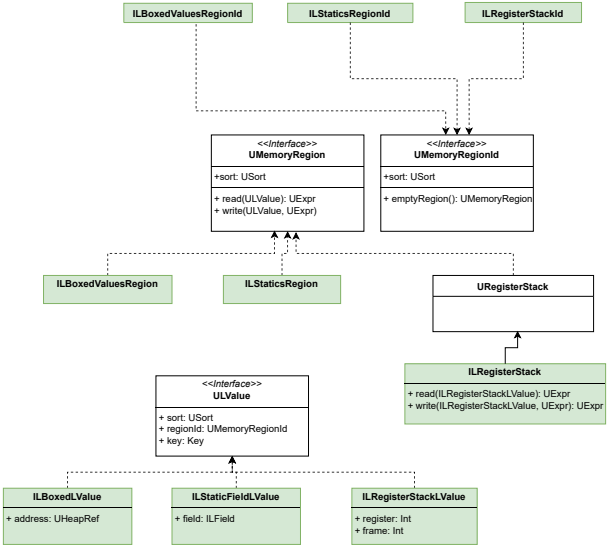
Результаты

Были достигнуты следующие результаты:

1. проведен обзор инструментов для генерации тестов и способов моделирования символьной памяти с небезопасным контекстом, для USVM-NET был выбран подход, основывающийся на подходе инструмента V# для генерации тестов и моделирования небезопасных операций
2. спроектирована архитектура символьной виртуальной машины USVM-NET
3. выполнена реализация символьной виртуальной машины USVM-NET без поддержки дженериков у символьных объектов, делегатов, многомерных массивов, Enum-ов, отслеживания границ циклов с символьным параметром
4. проведено интеграционное тестирование символьной машины

Архитектура памяти USVM-NET

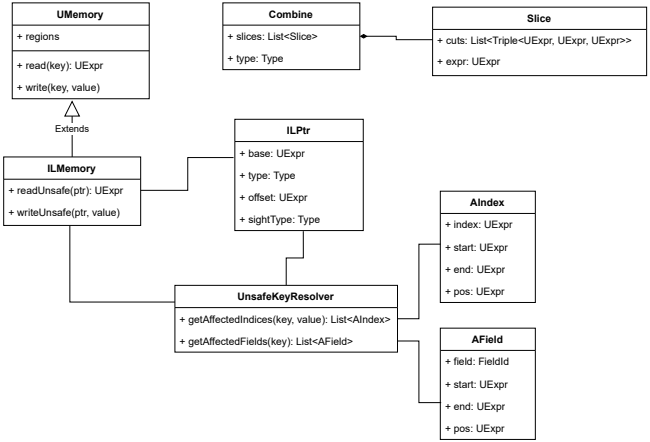
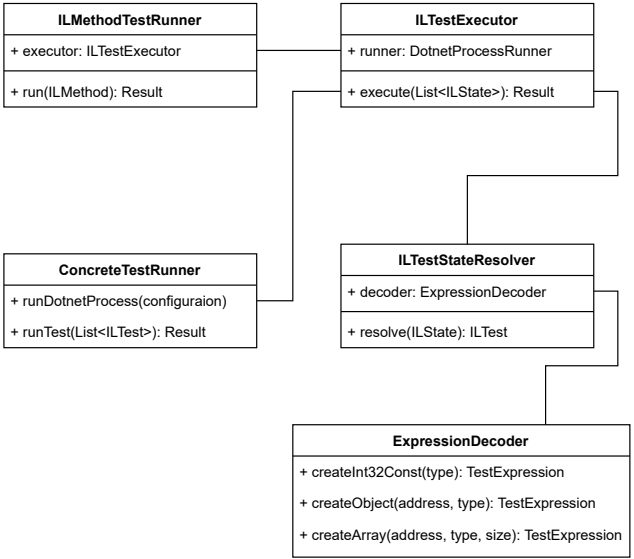
Безопасный контекст



Архитектура памяти USVM-NET

Небезопасный контекст

- основывается на подходе V#: доступ по небезопасным ключам сводится к безопасным ключам
- для кодирования используются теория бит-векторов, введены вспомогательные выражения **Slice** и **Combine**



Система генерации тестов