Модель символьной памяти .NET с поддержкой реинтерпретаций

Михаил Костицын
группа 471
научный руководитель: ст. преп. Д. А. Мордвинов

Санкт-Петербургский государственный университет
Кафедра системного программирования

11 июня 2019 г.

Символьное исполнение

Символьное исполнение — техника, которая позволяет моделировать исполнение программного кода в условиях неопределённости входных данных.

Пример

```
private unsafe static bool EqualsHelper(String strA, String strB) {
    int length = strA.Length;

    fixed (char* a = &strA.m_firstChar, b = &strB.m_firstChar) {
        while (length > 0) {
            int iA = *(int*)a;
            int iB = *(int*)b;
            if (iA != iB) break;
            a += 2; b += 2; length -= 2;
        }
        return (length <= 0);
    }
}
```

SMT-решатели

SMT-решатели — инструменты для автоматизированной проверки выполнимости логических формул в теориях.
Примеры теорий:

- Теория линейной арифметики
- Теория неинтерпретированных функций
- Теория массивов

Особенности памяти в платформе .NET

- Динамическая память
- Контексты исполнения:
  - Безопасный контекст — низкоуровневыми операциями с памятью управляет runtime
  - Небезопасный контекст — пользователю доступны операции с указателями

Реинтерпретация данных

Реинтерпретация данных — рассмотрение области памяти со структурой, отличной от той, с которой она задавалась.

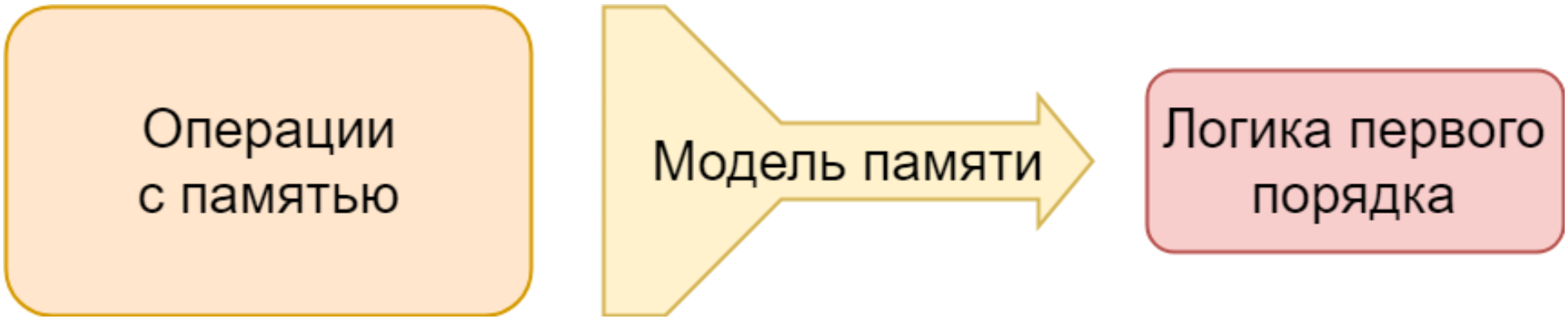
Пример

```
private unsafe static bool EqualsHelper(String strA, String strB) {
    int length = strA.Length;

    fixed (char* a = &strA.m_firstChar, b = &strB.m_firstChar) {
        while (length > 0) {
            int iA = *(int*)a;
            int iB = *(int*)b;
            if (iA != iB) break;
            a += 2; b += 2; length -= 2;
        }
        return (length <= 0);
    }
}
```

Модель памяти

Модель памяти — метод моделирования операций с памятью программы в логике первого порядка.



Цель работы

Целью данной работы является создание символьной модели памяти для платформы .NET с поддержкой реинтерпретации данных для верификации .NET-программ, основанной на SMT-решателях

Постановка задачи

- Выполнить обзор литературы о моделировании операций с памятью при помощи логики первого порядка, рассмотреть существующие анализаторы программ, поддерживающие реинтерпретацию данных
- Создать модель памяти для платформы .NET с поддержкой реинтерпретации данных
- Реализовать созданную модель памяти в символьной виртуальной машине $\mathrm{V}\#$
- Провести функциональное тестирование полученного решения

Существующие модели памяти

- Модели для высокоуровневого анализа памяти
  - \+ Поддержка произвольных свойств рекурсивных структур данных
  - – Отсутствие поддержки массивов, адресной арифметики, приведения типов указателей

Существующие модели памяти

- Модели для высокоуровневого анализа памяти
  - $+$ Поддержка произвольных свойств рекурсивных структур данных
  - $-$ Отсутствие поддержки массивов, адресной арифметики, приведения типов указателей
  - LISBQ
    - $\pm$ Анализ на основе аннотаций пользователя

Существующие модели памяти

- Модели для низкоуровневого анализа памяти
  - \+ Поддержка массивов, адресной арифметики, приведения типов указателей
  - – Отсутствие поддержки произвольных свойств рекурсивных структур данных

Существующие модели памяти

- Модели для низкоуровневого анализа памяти
  - \+ Поддержка массивов, адресной арифметики, приведения типов указателей
  - − Отсутствие поддержки произвольных свойств рекурсивных структур данных
  - Модель памяти на основе анализа алиасов
    - − Отсутствие поддержки произвольных свойств областей памяти заранее не ограниченного размера
    - ± Упрощение свойств памяти анализируемой программы

Существующие модели памяти

- Модели для низкоуровневого анализа памяти
  - $+$ Поддержка массивов, адресной арифметики, приведения типов указателей
  - $-$ Отсутствие поддержки произвольных свойств рекурсивных структур данных
  - $\pm$ Анализ на основе аннотаций пользователя

Существующие модели памяти

- Модели для низкоуровневого анализа памяти
  - \+ Поддержка массивов, адресной арифметики, приведения типов указателей
  - − Отсутствие поддержки произвольных свойств рекурсивных структур данных
  - ± Анализ на основе аннотаций пользователя
  - Типизированная модель памяти
    - \+ Сужение области возможных локаций благодаря типизации

Существующие модели памяти

- Модели для низкоуровневого анализа памяти
  - \+ Поддержка массивов, адресной арифметики, приведения типов указателей
  - − Отсутствие поддержки произвольных свойств рекурсивных структур данных
  - ± Анализ на основе аннотаций пользователя
  - Типизированная модель памяти
    - \+ Сужение области возможных локаций благодаря типизации
  - Модель памяти Бурсталла-Борната
    - \+ Раздельное представление состояния отдельных компонентов составных объектов

Существующие модели памяти

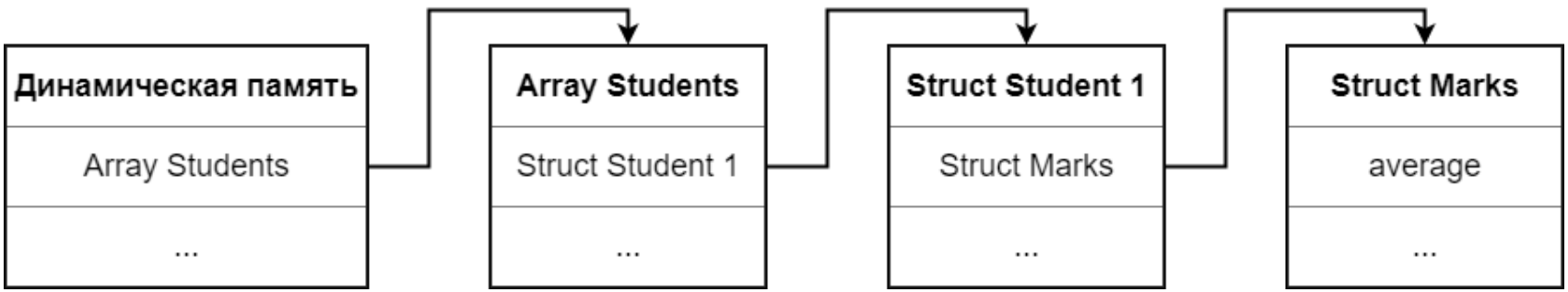
- Модели для низкоуровневого анализа памяти
  - \+ Поддержка массивов, адресной арифметики, приведения типов указателей
  - − Отсутствие поддержки произвольных свойств рекурсивных структур данных
  - ± Анализ на основе аннотаций пользователя
  - Типизированная модель памяти
    - \+ Сужение области возможных локаций благодаря типизации
  - Модель памяти Бурсталла-Борната
    - \+ Раздельное представление состояния отдельных компонентов составных объектов
  - Модель памяти с регионами
    - \+ Сужение области возможных локаций благодаря введению непересекающихся областей памяти

Модель памяти

- Иерархичность

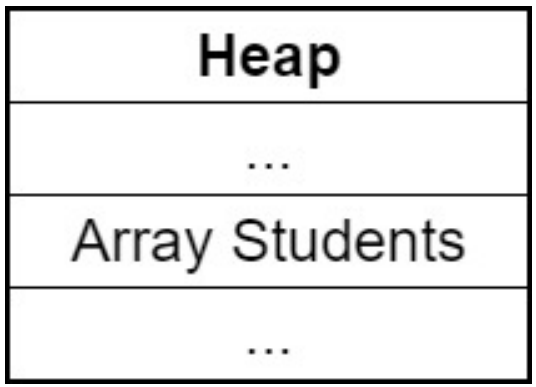
Пример

```
struct Marks
{
    private byte average;
    private byte exam;
}
struct Student
{
    private string name;
    private Marks marks;
}
...
Student[] Students = new Student[24];
```



Модель памяти

- Операции:
  - **alloc**
  - read
  - write



Пример

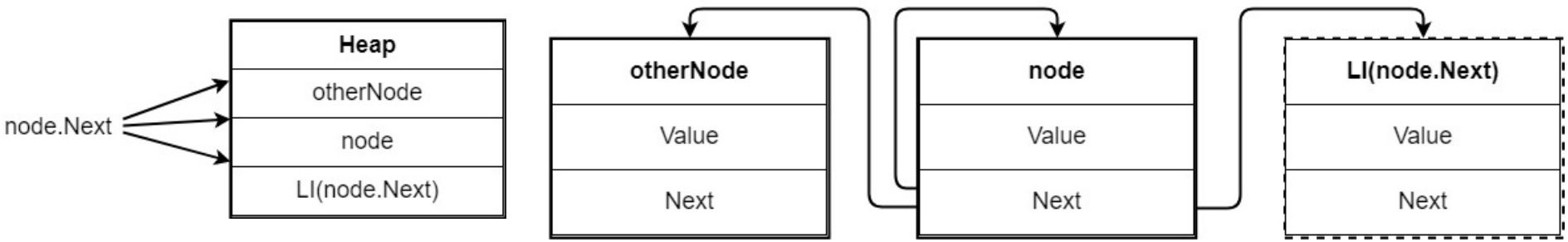
```
Student[] Students = new Student[24];
```

Модель памяти

- Операции:
  - alloc
  - **read**
  - **write**

Пример

```
static int TakeNext(LinkedListNode<int> node)
{
    return node.Next.Value;
}
```

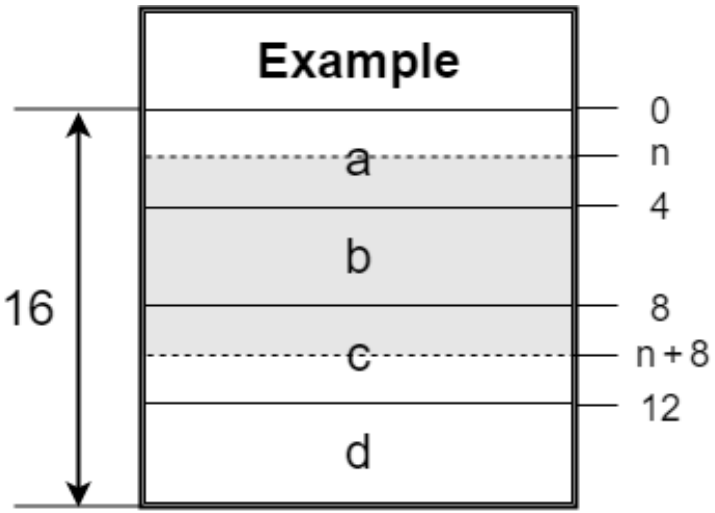


Модель памяти

- Реинтерпретация данных:
  - **Полей структуры или класса**
  - Элементов массива

Пример

```
struct Example
{
    public int a = 0xdead;
    public int b = 0xbeef;
    public int c = 0xcafe;
    public int d = 0xbabe;
}
...
Example ex = new Example();
int* ptr = &example.a + n;
long* intPtr = (long*)ptr;
long value = *ptr;
```



Модель памяти

$$LI(x) = \begin{cases} \texttt{STRUCT}(\epsilon), & isClassOrStruct(x) \\ \texttt{ARRAY}(\texttt{SYMBOL}, \epsilon, \epsilon, \epsilon), & isArray(x) \\ \texttt{SYMBOL}, & isSimpleType(x) \end{cases}$$

$$readStack(\varsigma, x, path) = \begin{cases} readTerm(\varsigma(x), path), & x \in dom(\sigma) \\ readTerm(LI(x), path), & x \notin dom(\sigma) \end{cases}$$

$$readHeap(\sigma, x, xs) = \texttt{UNION}\Big(\{\langle x = l, readTerm(\sigma(l), xs)\rangle \mid l \in dom(\sigma)\} \cup \langle \bigwedge_{l \in dom(\sigma)} x \neq l, readTerm(LI(x), xs)\rangle\Big)$$

$$readTerm(t, [x_{\mathbf{1}}; x_n]) = \begin{cases} t, & n = 0 \\ readHeap(get_\rho(t), y, [x_{\mathbf{2}}; x_n]), & n \neq 0 \wedge x_{\mathbf{1}} = \rho(y) \end{cases}$$

$$read(M, ref) = \begin{cases} \texttt{NRE}, & ref = \texttt{Ref}(\langle \texttt{Null}, xs\rangle) \\ myread_\xi(\xi(M), x, xs), & ref = \texttt{REF}(\langle \xi\texttt{Loc}(x), xs\rangle) \end{cases}$$

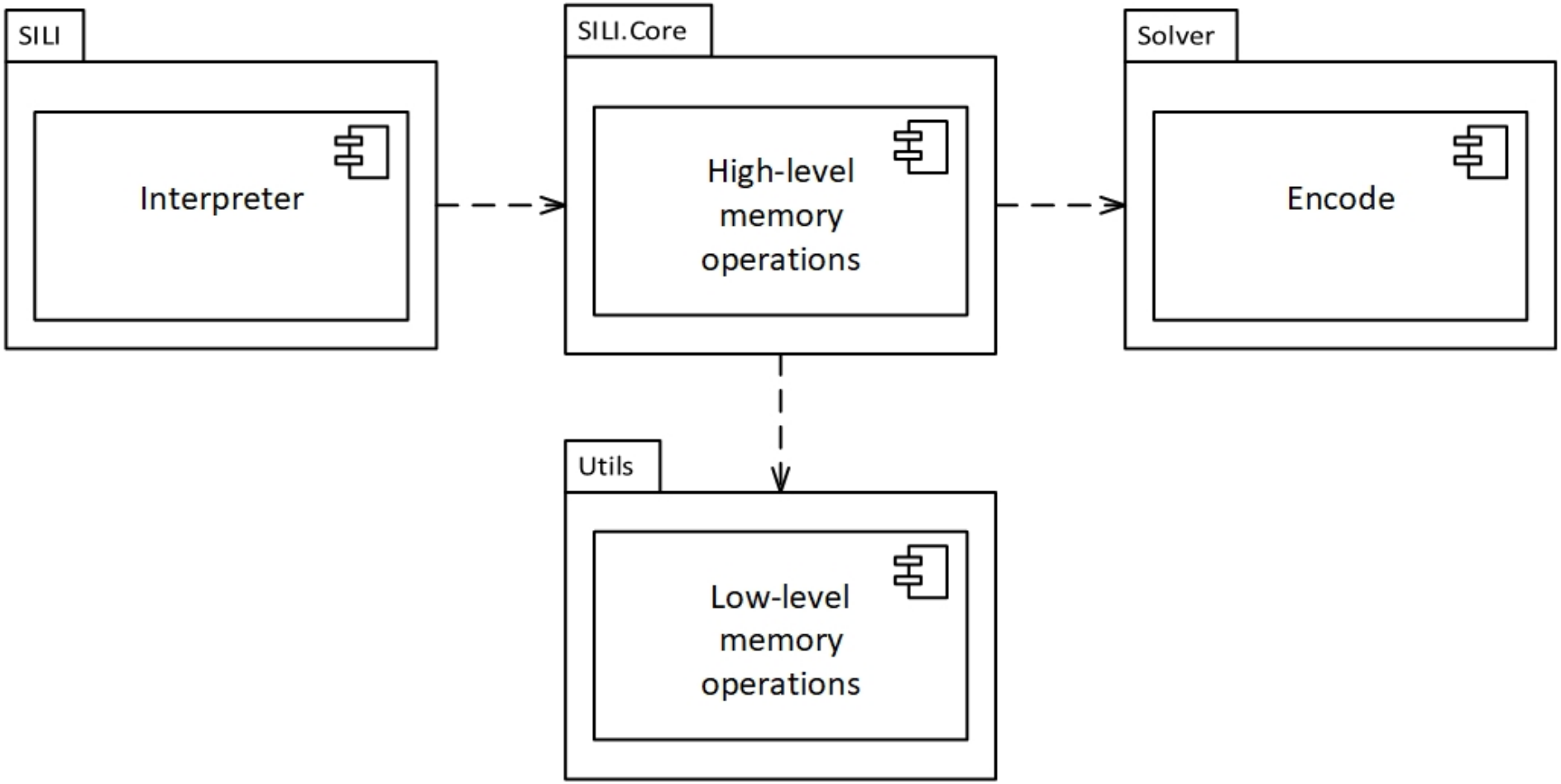
$$ptrRead(M, p) = \begin{cases} \texttt{UB}, & p = \texttt{PTR}(\langle \texttt{Null}, xs\rangle, offset) \\ readPtrTerm(read(M, LB), byte, ptrSize), & p = \texttt{PTR}(fql, offset) \end{cases}$$

$$readPtrTerm(t, byte, ptrSize) = \begin{cases} ptrStructRead(fields, byte, size, ptrSize), & t = \texttt{STRUCT}(fields) \\ ptrArrayRead(e, byte, esize, size, ptrSize), & t = \texttt{ARRAY}(d, e, lb, l) \\ ptrSimpleRead(t, byte, size, ptrSize), & isSimpleType(t) \end{cases}$$

$$ptrStructRead(fields, byte, size, ptrSize) = \texttt{UNION}\Big(\langle byte < 0 \vee byte > size - ptrSize, \texttt{UB}\rangle \cup \{\langle byte = b, readStructBytes(fields, b, size, ptrSize)\rangle \mid b \in [0; size - ptrSize]\}\Big)$$

$$ptrArrayRead(elems, byte, esize, size, ptrSize) = ite(byte < 0 \vee byte + ptrSize > size, \texttt{UB}, IntoArray(elems, ind, delta, esize, ptrSize))$$

Архитектура подсистемы



Тестирование функциональности

- Для тестирования функциональности системы были написаны тесты на языке C#, которые были исполнены в символьной виртуальной машине V#

Тестирование функциональности

Количественные характеристики тестов			
Название тестового набора	Количество тестов в наборе	Количество успешно пройденных тестов	Количество инструкций CIL
Strings	15	15	219
Unsafe	16	16	312
Classes and structs	10	10	316
Arrays	11	11	3619
Lists	6	6	1207
Всего	48	48	5673

Таблица: Результаты тестирования

Результаты

- Выполнен обзор следующих моделей памяти: модель на основе анализа алиасов, типизированная модель памяти, модель памяти Бурсталла-Борната, модель с регионами, LISBQ; и обзор анализаторов, поддерживающих реинтерпретацию данных
- Создана иерархическая модель памяти .NET с поддержкой реинтерпретации данных на основе теории битовых векторов и линейной арифметики
- Созданная модель памяти реализована внутри проекта V# на языке F#
- Проведено тестирование функциональности полученного решения