Композициональная верификация программ с динамической памятью на основе дизъюнктов Хорна

Юрий Костюков
группа 471
научный руководитель: ст. преп. Д. А. Мордвинов

Санкт-Петербургский государственный университет
Кафедра системного программирования

11 июня 2019 г.

Обзор

- композициональная
- **верификация**
- **программ с динамической памятью**
- на основе дизъюнктов Хорна

Трасса с ошибкой или **безопасный инвариант**

Обзор

- композициональная
- **верификация**
- **программ с динамической памятью**
- на основе дизъюнктов Хорна

Трасса с ошибкой или **безопасный инвариант**

Подходы (неточные):

- аппроксимация *снизу* — не все поведения программы
  - Например, раскрутка циклов: `for (int i = 0; i < 100; i++)`
- аппроксимация *сверху* — упрощение программы
  - Абстрактный домен: $\mathbb{Z} \rightsquigarrow \{+, -, 0\}$

$\Longrightarrow$ Нужен *точный* анализ

Обзор

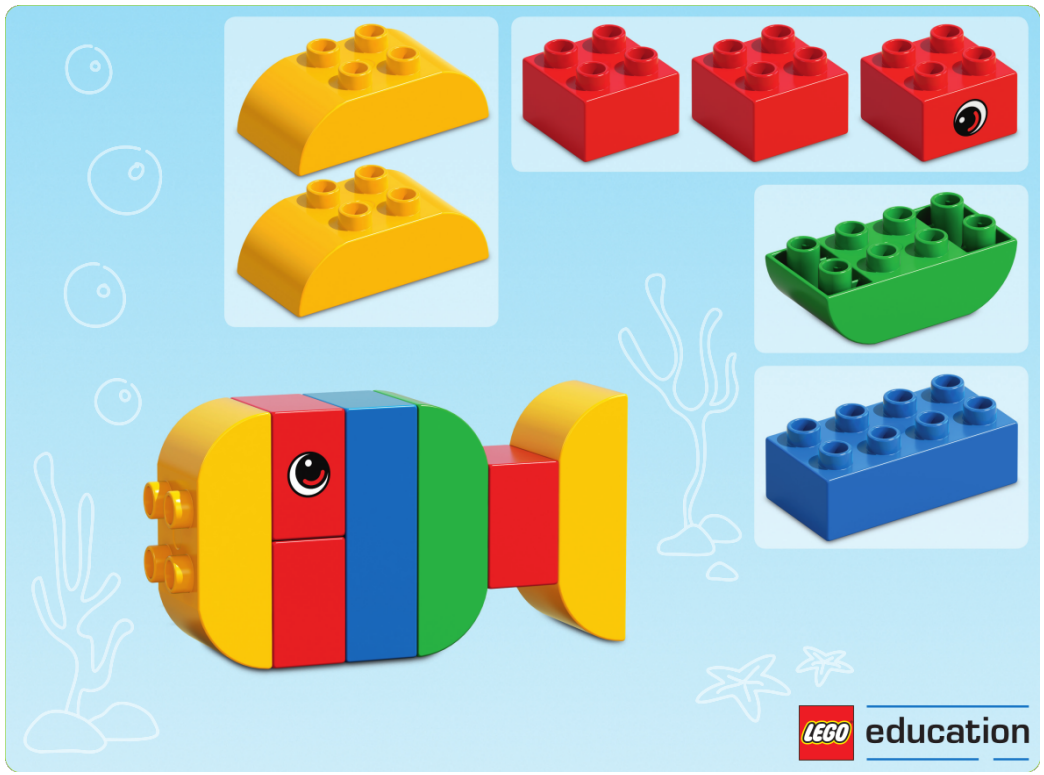
- композициональная
- верификация
- программ с динамической памятью
- **на основе дизъюнктов Хорна**

Система ограниченных дизъюнктов Хорна

$$xs = nil \wedge l = 0 \implies len(xs, l)$$
$$xs = cons(x, xs') \wedge l = l' + 1 \wedge len(xs', l') \implies len(xs, l)$$
$$xs = ys \wedge l_1 \neq l_2 \wedge len(xs, l_1) \wedge len(ys, l_2) \implies \bot$$

Обзор

- **композициональная**
- верификация
- программ с динамической памятью
- на основе дизъюнктов Хорна

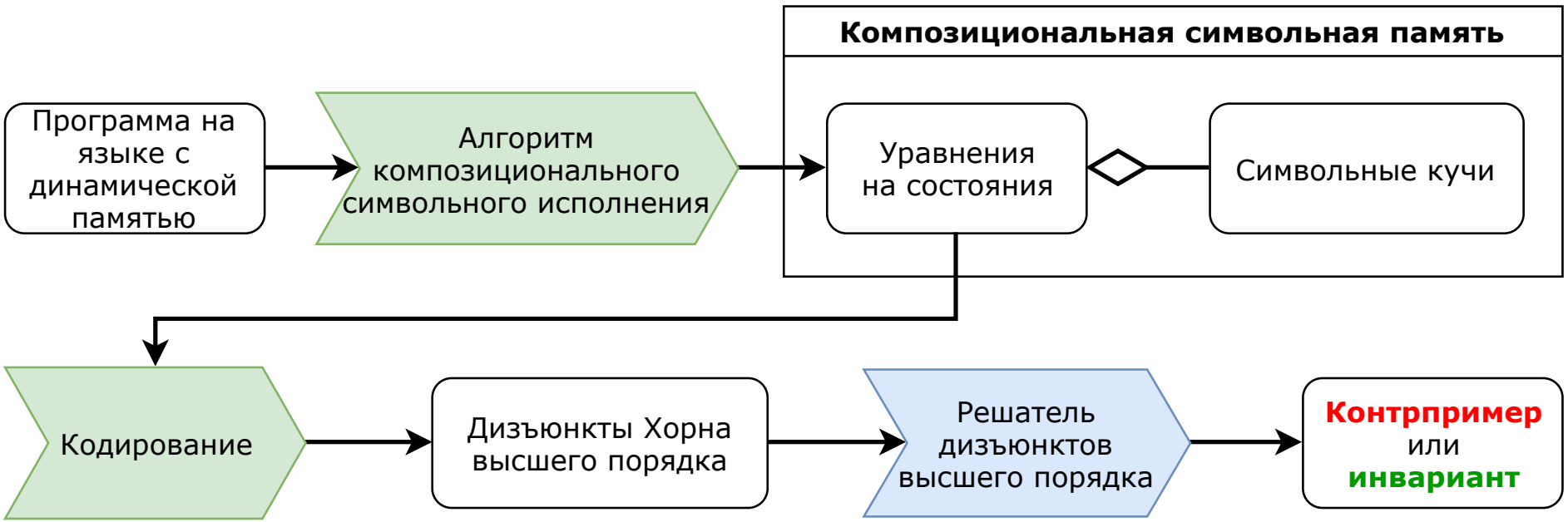


Цель работы

Разработать подход к автоматической композициональной верификации программ с динамической памятью. Задачи:

- Предложить подход к верификации программ с динамической памятью
- Доказать корректность предложенного подхода
- Реализовать подход
- Провести апробацию подхода

Подход


Композициональная символьная память
Программа на языке с динамической памятью
Алгоритм композиционального символьного исполнения
Уравнения на состояния
Символьные кучи
Кодирование
Дизъюнкты Хорна высшего порядка
Решатель дизъюнктов высшего порядка
Контрпример
или
инвариант

Композициональная символьная память: кучи

$$\texttt{term} ::= \texttt{leaf} \mid \texttt{op(term}^*\texttt{)} \mid \mathit{LI}(\mathit{loc}) \mid \mathit{union}\big(\langle \mathit{guard}, \mathit{term}\rangle^*\big)$$

Определение

Символьная куча — это отображение $\sigma : \mathit{loc} \to \mathit{term}$

Пример: композиция

```
class Node {
    int Key;
    Node Tail;
    Node(int k, Node t) {
        Key = k;
        Tail = t;
    }
}

void inc(Node node) {
    if (node != null) {
        node.Key = node.Key + 1;
    }
}
```

inc: $\{x.Key \mapsto LI(x.Key) + 1\}$

Пример: композиция

```
class Node {
    int Key;
    Node Tail;
    Node(int k, Node t) {
        Key = k;
        Tail = t;
    }
}

void inc(Node node) {
    if (node != null) {
        node.Key = node.Key + 1;
    }
}
```

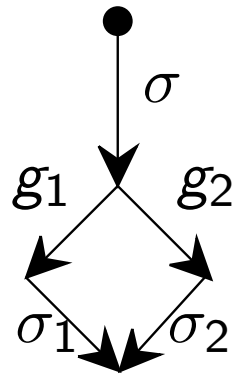
```
void Main() {
    Node x = new Node(42, null);
    inc(x);
}
```

inc: $\{x.Key \mapsto LI(x.Key) + 1\}$

Main: $\{x.Key \mapsto 42\} \circ \{x.Key \mapsto LI(x.Key) + 1\} = \{x.Key \mapsto 42 + 1\}$

Пример: merge

```
1  void SwapFirst2(Node x) {
2      if (x != null) {
3          int k = x.Key;
4          if (x.Tail != null) {
5              x.Key = x.Tail.Key;
6              x.Tail.Key = k;
7          } else {}
8      }
9  }
```



$\sigma \circ merge(\langle g_1, \sigma_1 \rangle, \langle g_2, \sigma_2 \rangle)$

- g_1: `x.Tail != null`
- g_2: $\neg$`(x.Tail != null)`
- σ: строка 3
- σ_1: строки 5-6
- σ_2: строка 7

Уравнения на состояния: пример

```
class Node {
    int Key;
    Node Tail;
    Node(int k, Node t) {
        Key = k;
        Tail = t;
    }
}
void inc(Node node) {
    if (node != null) {
        node.Key += 1;
        inc(node.Tail);
    }
}
```

```
void Main() {
    Node c = new Node(30, null);
    Node b = new Node(20, c);
    Node a = new Node(10, b);
    inc(a);
    Console.WriteLine(c.Key);
}
```

- $Rec(inc) = Merge(\langle LI(node) = 0, \epsilon \rangle, \langle LI(node) \neq 0, h_1 \circ Rec(inc) \rangle)$
- $App(Main) = h_2 \circ Rec(inc)$
- h_1 — куча, соответствующая инструкции `node.Key += 1;`
- h_2 — куча, соответствующая инициализации списка `(a, b, c)`

Уравнения на состояния: кодирование

```
class Node {
    int Key;
    Node Tail;
    Node(int k, Node t) {
        Key = k;
        Tail = t;
    }
}
```

```
data Node =  Key | Tail
type Addr = Int
type Loc = (Addr, Node)
ub _ = error "Undefined behaviour"
```

Уравнения на состояния: кодирование

```
void Main() {
    Node c = new Node(30, null);
    Node b = new Node(20, c);
    Node a = new Node(10, b);
    inc(a);
    Console.WriteLine(c.Key);
}
```

```
h2_int :: (Loc -> Int) -> Loc -> Int
h2_int _ (1, Key) = 10
h2_int _ (2, Key) = 20
h2_int _ (3, Key) = 30
h2_int ctx loc = ctx loc

h2_ptr :: (Loc -> Addr) -> Loc -> Addr
h2_ptr _ (1, Tail) = 2
h2_ptr _ (2, Tail) = 3
h2_ptr _ (3, Tail) = 0
h2_ptr ctx loc = ctx loc

main = print $ show $ inc_int (h2_int ub) (h2_ptr ub) 1 (3, Key)
```

Уравнения на состояния: кодирование

```
void inc(Node node) {
    if (node != null) {
        node.Key = node.Key + 1;
        inc(node.Tail);
    }
}
```

```
h1_int :: (Loc -> Int) -> Addr -> Loc -> Int
h1_int ctx node loc@(addr, Key) | addr == node = ctx loc + 1
h1_int ctx _ loc = ctx loc
```

Уравнения на состояния: кодирование

```
void inc(Node node) {
    if (node != null) {
        node.Key = node.Key + 1;
        inc(node.Tail);
    }
}
```

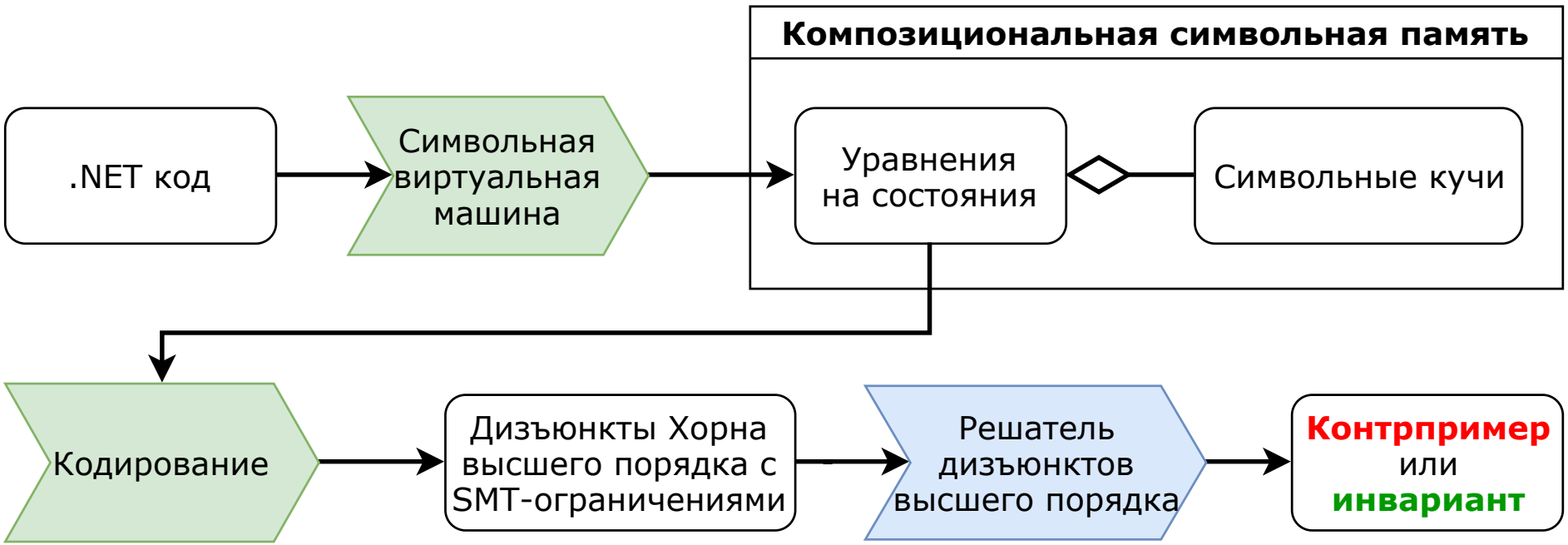
$$Rec(inc) = Merge(\langle LI(node) = 0, \epsilon \rangle, \langle LI(node) \neq 0, h_1 \circ Rec(inc) \rangle)$$

```
h1_int :: (Loc -> Int) -> Addr -> Loc -> Int
h1_int ctx node loc@(addr, Key) | addr == node = ctx loc + 1
h1_int ctx _ loc = ctx loc

inc_int :: (Loc -> Int) -> (Loc -> Addr) -> Addr -> Loc -> Int
inc_int ctx_int ctx_ptr 0 loc = ctx_int loc
inc_int ctx_int ctx_ptr node loc =
    let tail = ctx_ptr (node, Tail) in
    inc_int (h1_int ctx_int node) ctx_ptr tail loc
```

Реализация

- Подход реализован в проекте V#
- Реализация на языке F#, 1700 строк кода
- Сторонний решатель: RType + Spacer



Апробация

Набор тестов	Число запросов в наборе	Число верно отвеченных запросов	Среднее время на запрос, мс
Нерекурсивные структуры данных	24	24	1136
Рекурсивные структуры данных	42	39	1968

Результаты

- Предложен подход на основе *композициональной символьной памяти*
- Доказаны свойства корректности предложенной модели памяти
  - Доказательства частично механизированы в интерактивной системе проверки доказательств Coq
- Подход реализован в проекте V#
- Реализовано кодирование уравнений на состояния в дизъюнкты Хорна высших порядков
- Проведена апробация подхода на наборе нетривиальных рекурсивных программ с изменяемыми рекурсивными структурами данных