Синтез моделей внешних вызовов для символьного исполнения

Зимин Григорий Александрович

Руководитель: доц. каф СП, к.т.н. Литвинов Ю.В.
Рецензент: разработчик ПО ООО «ИнтеллиДжей Лабс» Мордвинов Д.А.

Санкт-Петербург, 2018

Корректность и надежность программ

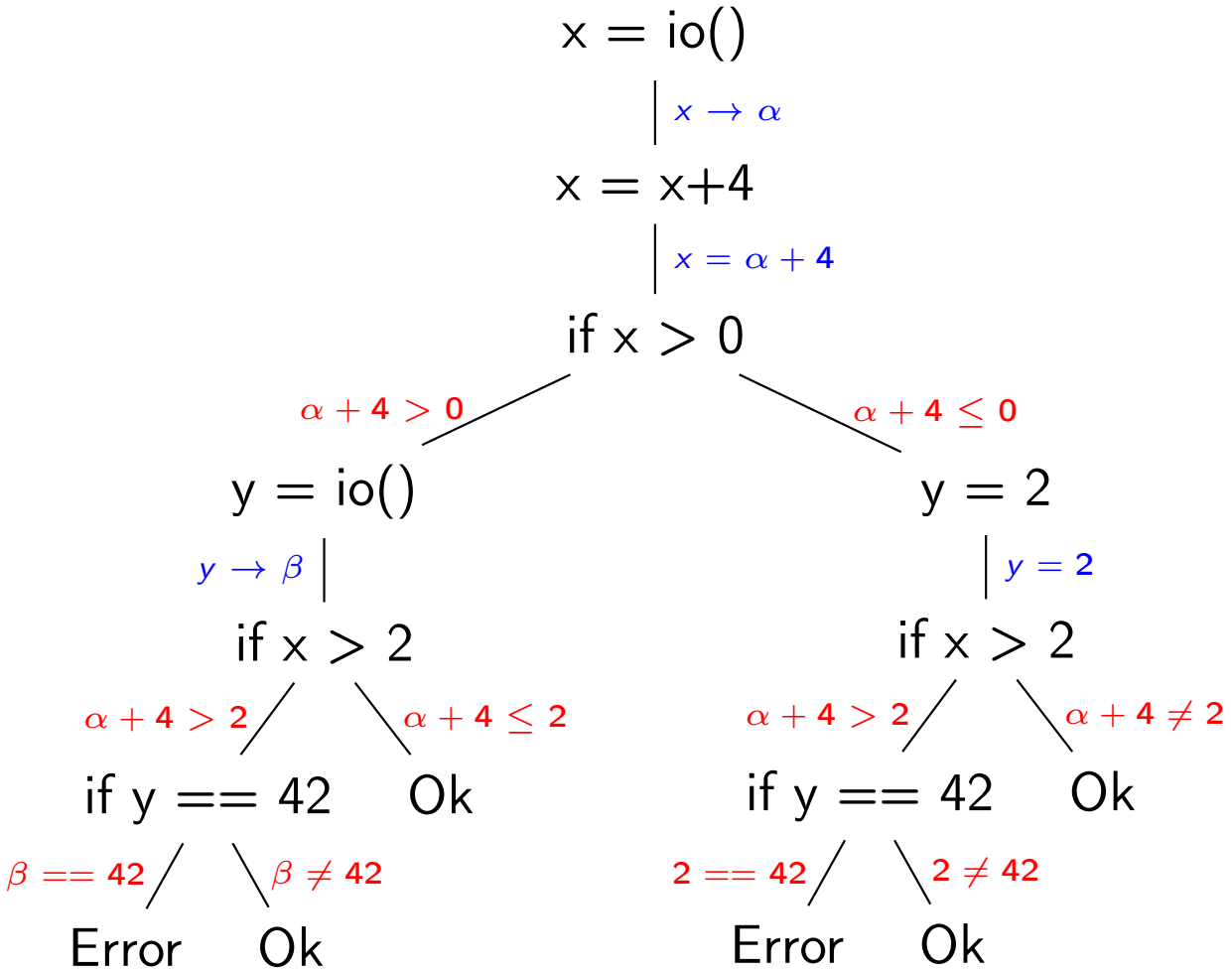
- Автоматический анализ программного кода
  - Символьное исполнение программ

Символьное исполнение (СИ)

```
x = io();
x = x + 4;

if (x > 0)
 y = io();
else
 y = 2;

if (x > 2)
 if (y == 42)
  fail("Error");
```



Фундаментальная проблема СИ

- Недоступные участки кода
  - Библиотечные функции
  - Системные вызовы
  - ...
- Традиционные способы решения
  - Вызовы на конкретных значениях
  - Модели, написанные вручную

Мотивация работы

Автоматический синтез моделей для библиотечных вызовов

Синтаксически-направленный синтез

Задача Syntax-Guided Synthesis (SyGuS)

Найти $f_{implementation} \in L$, такую что $\phi[f / f_{implementation}]$ выполнима по модулю теории T

- ϕ — спецификация искомой функции f в виде формулы на языке логики первого порядка
- L — множество выражений, построенное по грамматике G

Победители соревнований SyGuS-comp

- `CVC4`
- `EUSolver`

Цель и задачи

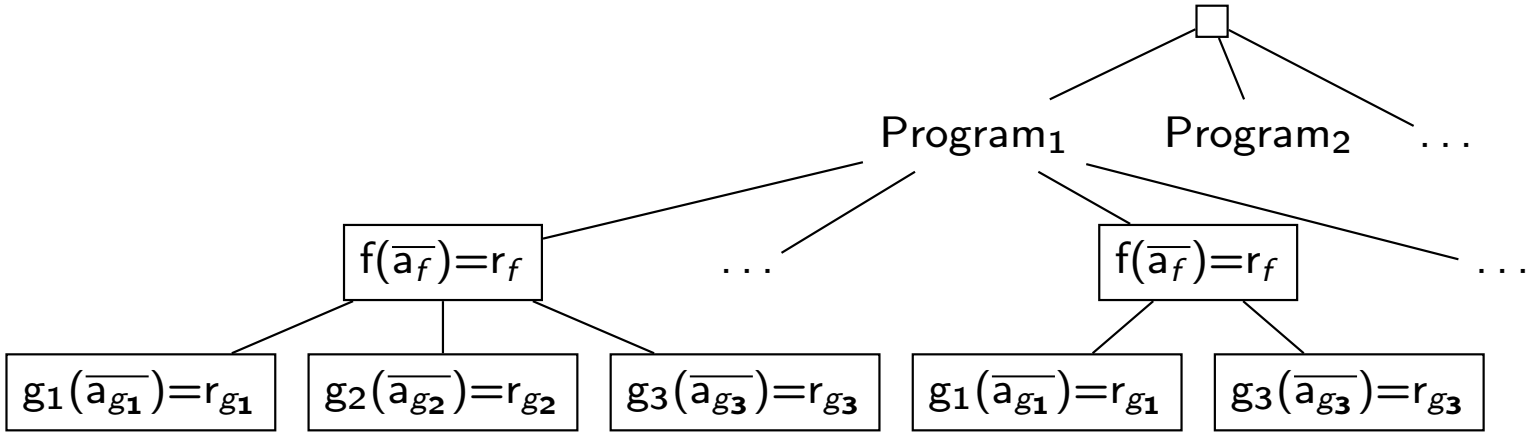
Создание прототипа инструмента для автоматического синтеза моделей внешних вызовов

1. Разработать метод сведения задачи синтеза моделей внешних вызовов к проблеме синтаксически-направленного синтеза
2. Создать прототип инструмента для автоматического синтеза моделей внешних вызовов
3. Провести апробацию инструмента с помощью решателей задачи SyGuS

Алгоритм синтеза

1. Сбор программ, их трассировка
2. Восстановление потока управления функций
3. Формирование задач для SyGuS-решателей
4. Решение задач решателями
5. Генерация кода по решенным задачам

Восстановление потока управления



Вид потока управления

- Задача о нахождении общей надстроки

```
1  | f(af) {
2  |       rg1 = g1(ag1);
3  |       if (condition)
4  |            rg2 = g2(ag2);
5  |
6  |
7  |       rg3 = g3(ag3);
8  |       rf = f(af, ri, statef);
9  |       effects
10 |       return rf;
11 | }
```

Пример императивных функций

Последовательность вызовов

```
write("hello␣");          // 6
read();                   // "hello "
write("world");           // 5
read();                   // "hello world"
```

Изменение сигнатуры

Результат как часть состояния

$$\mathtt{F} : \mathtt{arguments} \times \mathtt{state} \to \mathtt{state}$$

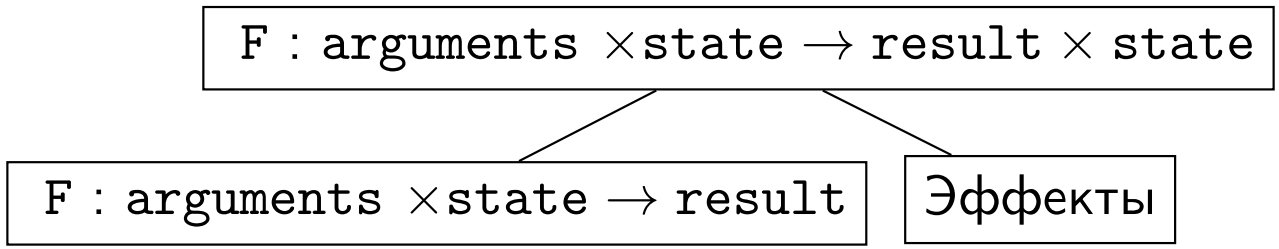
Последовательное применение функций

$$\mathtt{F}_i(\mathtt{arguments}, \ldots \mathtt{F}_j(\mathtt{arguments}, \mathtt{state}) \ldots) = \mathtt{state}$$

Запись для примера

```
write("hello␣", rRes, wRes, stateS);
read(write("hello␣", rRes, wRes, stateS));
write("world", read(write("hello␣", rRes, wRes, stateS)));
read(write("world", read(write("hello␣", rRes, wRes, stateS))));
```

«Декомпозиция функции»



Функция-эффект

$$F_i^j : args_i \times res_i \times state_j \to state_j$$

Порядок вызовов

```
write("hello␣");
read();
write("world");
read();
```

Функция read

```
read(effect^read_write("hello␣", 6, stateS));
read(
 effect^read_write("world", 5,
   effect^read_write("hello␣", 6, stateS)
));
```

Оракулы

- Оракул зависимостей между функциями
  - Цель — отсечение заведомо независимых функций
  - Способ — аргументы-идентификаторы
- Оракул состояния функции
  - Цель — определение типов элементов состояния
  - Способ — комбинирование типов аргументов и результатов функций

Формирование задач в формате SyGuS

Функция read

```
read(effect^{read}_{write}("hello␣", 6, stateS));
read(effect^{read}_{write}("world", 5, effect^{read}_{write}("hello␣", 6, stateS)));
```

Отрывок записи задачи для функции read

```
(synth-fun E^r_w ((WA String) (result Int) (RS String) (RS_Upd String)) Bool (...))
(synth-fun read ((RS String)) String (...))
...
(constraint
(= true (and
 (=> (and (init_RS RS_0) (E^r_w "hello␣" 6 RS_0 RS_1))
     (= (read RS_1) "hello␣"))
 (=> (and (init_RS RS_0) (E^r_w "hello␣" 6 RS_0 RS_1) (E^r_w "world" 5 RS_1 RS_2))
     (= (read RS_2) "hello␣world"))))
)
```

Генерация кода по решенным задачам

Результаты синтеза в формате SyGuS

```
(define-fun init_WS ((st String)) Bool (= st ""))
(define-fun write ((WA String) (WS String)) Int (str.len WA))
(define-fun E_r^w ((result String) (WS String) (WS_Upd String)) Bool (= WS_Upd WS))

(define-fun init_RS ((stateVar String)) Bool (= stateVar ""))
(define-fun read ((RS String)) String RS)
(define-fun E_w^r ((WA String) (result Int) (RS String) (RS_Upd String)) Bool
 (= RS_Upd (str.++ RS WA)))
```

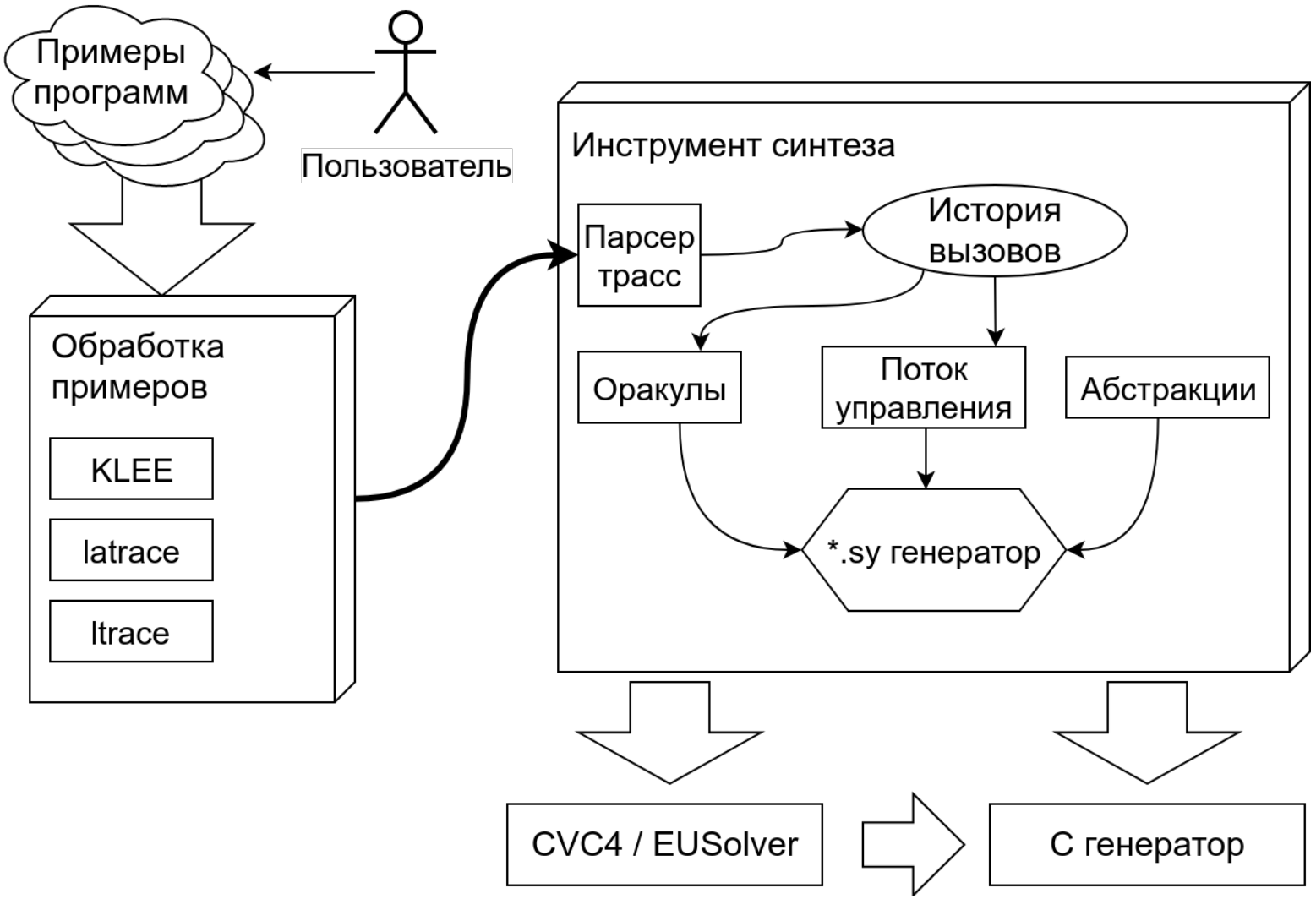
Результат синтеза на C (1/2)

```
char writeState[256] = "";
char readState[256] = "";
char *read() {
  char *result;
  result = readState;
  return result;
}
```

Результат синтеза на C (2/2)

```
int write(char *str) {
  int result;
  result = strlen(str);
  // effects
  strcat(readState, str);
  return result;
}
```

Общая структура инструмента



Эксперименты

- Синтетические трассы
  - Функции без состояний
    - Простые линейные функции
    - Функции `min3`/`max3`, реализованные как 3 вызова `min2`/`max2`
  - Функции с состояниями
    - Упрощенные версии функций `write`, `read`, `open`, `close`
    - Различные линейные функции с ветвлением внутри
- Функции из заголовочного файла `stdio.h`
  - Собрано 43 примера, покрывающих работу 17 функций
  - По ним сформировано 30 + 86 задач SyGuS
  - Завершились задачи для функций
    - `fseek`, `rewind`, `strlen`

Результаты работы

- Разработан метод синтеза моделей внешних вызовов по трассам программ. Представленный метод позволяет свести задачу синтеза моделей императивных функций к проблеме синтаксически-направленного синтеза
- Создан прототип инструмента, автоматизированно выполняющий сбор трасс из набора примеров и реализующий описанный метод
- Проведена апробация подхода на синтетических трассах и реальных примерах кода с помощью решателей `CVC4` и `EUSolver`