

Синтез программ по спецификациям с
множественными вызовами

Мишенев Вадим Сергеевич
Руководитель: д. т. н., профессор Кознов Д. В.
Консультант: Университет штата Флорида

Консултант: Университет штата Флорида

Рецензент: ООО «ИнтеллиДжей Лабс»

разработчик ПО Косарев Д. А.

9 июня 2020 г.

Здравствуйте! Тема моей работы: Синтез программ по спецификациям с множественными вызовами.

Автор: Мишенев Вадим

Научный руководитель: Кознов Дмитрий Владимирович

Определение

Синтез программ — задача автоматического создания программ на некотором языке программирования, которые удовлетворяют намерениям пользователя, выраженным в форме ограничений.

```
=LOWER(LEFT(B2,1))&"."&LOWER(A2)&"@"&LOWER(C2)&".com"
```

	A	B	C	D
1	Last name	First name	Domain	Email address
2	Anderson	Ronnie	Gmail	r.anderson@gmail.com
3	Boone	Tom	Hotmail	t.boone@hotmail.com
4	Brook	Sally	Outlook	s.brook@outlook.com
5	Hill	Jeremy	Gmail	j.hill@gmail.com
6	Waldau	Mattias	Hotmail	m.waldau@hotmail.com

Синтез программ

└ Понятие синтеза программ

Понятие синтеза программ

Определение

Синтез программ — задача автоматического создания программ на некотором языке программирования, которые удовлетворяют намерениям пользователя, выраженным в форме ограничений.

	A	B	C	D
1	Last name	First name	Domain	Email address
2	Anderson	Ronnie	Gmail	r.anderson@gmail.com
3	Boone	Tom	Hotmail	t.boone@hotmail.com
4	Brook	Sally	Outlook	s.brook@outlook.com
5	Hill	Jeremy	Gmail	j.hill@gmail.com
6	Waldau	Mattias	Hotmail	m.waldau@hotmail.com

Синтез программ является задачей автоматического создания программ на некотором языке программирования, которые удовлетворяют намерениям пользователя, выраженным в форме ограничений. Например, синтез программ используется при экстраполяции ячеек в Excel, где по примерам пользователя синтезируется программа для обработки строк.

Другой пример, синтез выигрышных стратегий в играх, которые используется при планировании движения роботов.

- Реактивных системах
- Формальная верификация, например, синтез моделей программ
- Программирование с помощью наброска программ (программ с "дырками")
- Простой способ описания небольших программ
- Синтез функциональных программ

```
1 generator int linexp(int x, int y) {
2     return ?? * x + ?? * y;
3 }
4                                     val (hrs, mns, scs) = choose((h: Int, m: Int, s: Int) =>
5                                     h * 3600 + m * 60 + s == totsec &&
6                                     0 ≤ h < 24 &&
7                                     0 ≤ m && m < 60 &&
8                                     0 ≤ s && s < 60)
9                                     harness void main(int x, int y) {
10                                     assert linexp(x, y) == x + x + y;
11                                     assert linexp(x, y) == x + y + y;
12 }
```

Синтез программ

└ Применение

Применение

- Реактивных системах
- Формальная верификация, например, синтез моделей программ
- Программирование с помощью наброска программы (программы с "дырками")
- Простой способ описания небольших программ
- Синтез функциональных программ

```
1 generator int linexp(int x, int y) {
2     return ?? * x + ?? * y;
3 }
4
5 harness void main(int x, int y) {
6     assert linexp(x, y) == x + x + y;
7     assert linexp(x, y) == x + y + y;
8 }
```

Кроме того, синтез программ находит широкое применение в реактивных системах, верификации, программирование с пропущенными выражениями и так далее.

Пример

$$\max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$$

2020-06-09

Мишенев Вадим Сергеевич (СПБГУ) Синтез программ 9 июня 2020 г. 4 / 13

Синтез программ

└─ Функциональный синтез

Функциональный синтез

Пример
 $\max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$

Среди известных подходов к синтезу программ существует эффективный и масштабируемый подход, называемый функциональным синтезом.

Рассмотрим пример такого синтеза. Пусть в качестве намерений пользователя есть некоторая логическая формула, которая связывает входные и выходные параметры программы.

Пример

$\max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$

```
int max(int x, int y) {  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

Синтез программ

└ Функциональный синтез

Функциональный синтез

Пример

$\max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$

```
int max(int x, int y) {  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

Тогда по такой формуле можно синтезировать программу на слайде.

Пример

$\exists \max \forall x, y. \max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$

```
int max(int x, int y) {  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```



$\max(x, y) = \text{ite}(x > y, x, y)$

Такую программу можно представить через формулу $\max(x, y) = \text{ite}(x > y, x, y)$ и рассматривать как функцию.

Задача функционального синтеза

Найти реализацию f для $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ в некоторой формальной теории

- $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ — спецификация

Синтез программ

└─ Задача функционального синтеза

Задача функционального синтеза

Задача функционального синтеза

Найти реализацию f для $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ в некоторой формальной теории

- $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ — спецификация

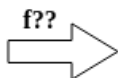
Таким образом, в рамках функционального синтеза синтезируемая программа рассматривается как нерекурсивная функция, а намерения пользователя выражаются в виде спецификации программы, т.е. логической формулы некоторой формальной теории.

Задача функционального синтеза

Найти реализацию f для $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ в некоторой формальной теории

- $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ — спецификация
- **Решение** — выражение f_{impl} , подстановка которого вместо f делает $\forall \bar{x}. \varphi[\bar{x}, f_{impl}/f]$ общезначимой в некоторой формальной теории
- Если f_{impl} не существует, то спецификация **нереализуема**

```
void foo(int x, int y) {  
    int z = f(x);  
    int w = f(y);  
    assert(z + w >= x - y);  
}
```



$\exists f. \forall x, y. f(x) + f(y) \geq x - y$

Синтез программ

Задача функционального синтеза

2020-06-09

Задача функционального синтеза

Задача функционального синтеза

Найти реализацию f для $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ в некоторой формальной теории

- $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ — спецификация
- **Решение** — выражение f_{impl} , подстановка которого вместо f делает $\forall \bar{x}. \varphi[\bar{x}, f_{impl}/f]$ общезначимой в некоторой формальной теории
- Если f_{impl} не существует, то спецификация **нереализуема**

```
void foo(int x, int y) {  
    int z = f(x);  
    int w = f(y);  
    assert(z + w >= x - y);  
}
```

$\exists f. \forall x, y. f(x) + f(y) \geq x - y$

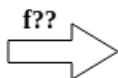
Тогда задача синтеза: найти реализация функции в виде терма теории с оператором ветвления (если-тогда-иначе — *ite*), подстановка которого в спецификацию делает её общезначимой в рассматриваемой теории. Если такого терма нет, то спецификация нереализуема. На слайде показан возможный способ генерации спецификации.

Задача функционального синтеза

Найти реализацию f для $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ в **целочисленной линейной арифметике**

- $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ — спецификация
- **Решение** — выражение f_{impl} , подстановка которого вместо f делает $\forall \bar{x}. \varphi[\bar{x}, f_{impl}/f]$ общезначимой в **целочисленной линейной арифметике**
- Если f_{impl} не существует, то спецификация **нереализуема**

```
void foo(int x, int y) {  
    int z = f(x);  
    int w = f(y);  
    assert(z + w >= x - y);  
}
```



$\exists f. \forall x, y. f(x) + f(y) \geq x - y$

Синтез программ

└─ Задача функционального синтеза

2020-06-09

Задача функционального синтеза

Задача функционального синтеза

Найти реализацию f для $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ в **целочисленной линейной арифметике**

• $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ — спецификация

• **Решение** — выражение f_{impl} , подстановка которого вместо f делает $\forall \bar{x}. \varphi[\bar{x}, f_{impl}/f]$ общезначимой в **целочисленной линейной арифметике**

• Если f_{impl} не существует, то спецификация **нереализуема**

```
void foo(int x, int y) {  
    int z = f(x);  
    int w = f(y);  
    assert(z + w >= x - y);  
}
```

$\Rightarrow \exists f. \forall x, y. f(x) + f(y) \geq x - y$

В моей работе будет рассматриваться только теория целочисленной линейной арифметики.

- **Спецификация с одиночным вызовом** — функция f входит всюду в спецификацию с одним и тем же набором аргументов

Пример

$$\max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$$

2020-06-09

Мишенев Вадим Сергеевич (СПбГУ) Синтез программ 9 июня 2020 г. 6 / 13

Синтез программ

Спецификации

• Спецификация с одиночным вызовом — функция f входит всюду в спецификацию с одним и тем же набором аргументов

Пример
 $\max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$

└ Спецификации

Спецификация называется с одиночным вызовом, если в неё синтезируемая функция встречается всюду только с одним и тем же фиксированным набором аргументов.

- Спецификация с одиночным вызовом — функция f входит всюду в спецификацию с одним и тем же набором аргументов

Пример

$$\max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$$

- Иначе спецификация с множественными вызовами

Пример

$$f(x + 1) > f(x)$$

Иначе с множественными , что является более общим случаем.

- Спецификация с одиночным вызовом — функция f входит всюду в спецификацию с одним и тем же набором аргументов

Пример

$$\max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$$

- Иначе спецификация с множественными вызовами

Пример

$$f(x+1) > f(x)$$

- Синтез по спецификациям с одиночным вызовом — хорошо изучен
- Синтез по спецификациям с множественными вызовами — алгоритмически неразрешим

2020-06-09

Мишенев Вадим Сергеевич (СПбГУ) Синтез программ 9 июня 2020 г. 6 / 13

Синтез программ

Спецификации

• Спецификация с одиночным вызовом — функция f входит всюду в спецификацию с одним и тем же набором аргументов

Пример

$$\max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$$

• Иначе спецификация с множественными вызовами

Пример

$$f(x+1) > f(x)$$

• Синтез по спецификациям с одиночным вызовом — хорошо изучен

• Синтез по спецификациям с множественными вызовами — алгоритмически неразрешим

Синтез по спецификациям с одиночным вызовом хорошо изучен, было предложено много подходов.

Напротив, синтез по спецификациям с множественными вызовами алгоритмически неразрешим даже для разрешимой теории линейной арифметики, и такой синтез является открытой проблемой.

Цель: Создание алгоритма синтеза программ по логическим спецификациям с множественными вызовами

Задачи:

- Провести анализ предметной области и существующих подходов к синтезу программ: функциональный синтез, синтаксически-управляемый синтез
- Разработать алгоритм для синтеза по спецификациям с одиночным вызовом
- Разработать алгоритм для синтеза по спецификациям с множественными вызовами
- Реализовать оба алгоритма в рамках одного программного инструмента
- Провести экспериментальное исследование разработанного решения

Цель моей работы: Создание алгоритма синтеза программ по логическим спецификациям с множественными вызовами.

Задачи представлены на слайде.

Основные задачи: разработка алгоритмов для синтеза по спецификациям с одиночными и множественными вызовами, их реализация и экспериментальное исследование.

Lazy but Effective Functional Synthesis

Gregory Petroskiy¹, Alex Gladkov², Andre Geras³

¹ Princeton University, Princeton, USA, {gpetroski,agladkov}@princeton.edu
² University of Toronto, Toronto, Canada, agladkov@cs.toronto.edu, ca

Abstract. We present a new technique for generating a functional implementation from a declarative specification formalized as a $\forall\exists$ formula in quantifier-free logic. We follow a simple scheme of generating a universal quantifier-free formula from the original formula by iteratively removing quantifiers. The method allows the quantifier body and predicate a constant number of iterations of a quantifier-free formula to be applied, and the result is a formula that can be used for further processing. The method can be used to generate a constant number of iterations of a quantifier-free formula to be applied, and the result is a formula that can be used for further processing. The method can be used to generate a constant number of iterations of a quantifier-free formula to be applied, and the result is a formula that can be used for further processing.

1 Introduction

The task of generating a function implementation from a specification of an input-output relation is naturally addressed by functional synthesis. While prior approaches have been proposed for functional synthesis [1,2,3,4,5,6,7], they all require a complex preprocessing step to transform the input formula into a form suitable for synthesis. In this paper, we propose a new technique for generating a function implementation from a specification of an input-output relation. The method allows the quantifier body and predicate a constant number of iterations of a quantifier-free formula to be applied, and the result is a formula that can be used for further processing. The method can be used to generate a constant number of iterations of a quantifier-free formula to be applied, and the result is a formula that can be used for further processing.

Использование модельных проекций (MBP)

$MBP_{\bar{y}}$ — бескванторная формула над \bar{y}

$$\exists \bar{x}.\psi(\bar{x}, \bar{y}) \equiv \bigvee_i MBP_{\bar{y}}(m_i, \psi)$$

Lazy and Effective
Functional Synthesis,
VMCAI 2019

2020-06-09

Синтез программ

Упрощение спецификаций

Упрощение спецификаций

Использование модельных проекций (MBP)

$MBP_{\bar{y}}$ — бескванторная формула над \bar{y}

$$\exists \bar{x}.\psi(\bar{x}, \bar{y}) \equiv \bigvee MBP_{\bar{y}}(m_i, \psi)$$

В 2019 году (на VMCAI) была опубликована статья про функциональный синтез на основе модельных проекциях, которая была взята за основу для разработки алгоритма синтеза.

Модельная проекция — современный подход, предложенный в 2015 году, о котором более подробно описано в отчёте.

Модельная проекция используется для упрощения спецификации и эффективного поиска ветвлений, которая позволяет рассматривать не целиком всю спецификацию, а только некоторую её часть.

Lazy but Effective Functional Synthesis

Gergely Petrichukov¹, Amr Ghallab², Aarti Gupta³

¹ Princeton University, Princeton, USA, gpetrich@princeton.edu
² University of Paris-Saclay, Nanterre, France, amr.ghallab@universite-paris-saclay.fr

Abstract. We present a new technique for generating a C++ implementation from a declarative specification formalized as a $\forall\exists$ formula in quantifier prefix form. We follow a simple pipeline of generating a relational network of existing library functions for the quantifier prefix, then a C++ implementation for the quantifier body and postcondition, and a verification condition for the body of a quantifier. Compared to other approaches, our method runs faster. Our results on the existing benchmarks show that we can check both safety and reachability as well as generate executable code. Our approach is implemented in a tool called *ABC-SYN*, and the results are on a set of existing synthesis benchmarks above previous.

1 Introduction

The task of generating a function implementation from a specification of an input-output relation is commonly addressed by *function synthesis*. While prior approaches have been proposed for functional languages [1,2,3,4,5,6,7], the problem is more difficult in imperative languages. Even generating a verification condition requires finding a suitable library function for the quantifier prefix, and then a C++ implementation for the quantifier body and postcondition. Our method runs faster. Our results on the existing benchmarks show that we can check both safety and reachability as well as generate executable code. Our approach is implemented in a tool called *ABC-SYN*, and the results are on a set of existing synthesis benchmarks above previous.

Function tasks are often formulated as quantified formulas. We consider formulas of the form $\forall \bar{x}. \exists \bar{y}. \psi(\bar{x}, \bar{y})$ for \bar{x}, \bar{y} being vectors of variables. A formula is satisfiable if a quantifier-free formula ϕ is satisfiable. A formula is unsatisfiable if a quantifier-free formula ϕ is unsatisfiable. A formula is satisfiable if a quantifier-free formula ϕ is satisfiable. A formula is unsatisfiable if a quantifier-free formula ϕ is unsatisfiable.

² Also see [8], which uses the same notation to denote multiple variables.

Lazy and Effective
Functional Synthesis,
VMCAI 2019

Использование модельных проекций (MBP)

$MBP_{\bar{y}}$ — бескванторная формула над \bar{y}

$$\exists \bar{x}. \psi(\bar{x}, \bar{y}) \equiv \bigvee_i MBP_{\bar{y}}(m_i, \psi)$$

Нереализуемые спецификации

$$f(x) > f(x-1) \wedge f(y) > f(y+1)$$

Для $x \mapsto 1, y \mapsto 0$:

$$f(1) > f(0) \wedge f(0) > f(1)$$

Мишнев Вадим Сергеевич (СПБГУ)

Синтез программ

9 июня 2020 г.

8 / 13

Синтез программ

└ Упрощение спецификаций

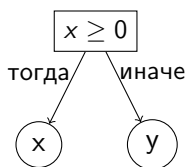
Упрощение спецификаций

Использование модельных проекций (MBP)
 $MBP_{\bar{y}}$ — бескванторная формула над \bar{y}
 $\exists \bar{x}. \psi(\bar{x}, \bar{y}) \equiv \bigvee_i MBP_{\bar{y}}(m_i, \psi)$

Нереализуемые спецификации

Для $x \mapsto 1, y \mapsto 0$:
 $f(1) > f(0) \wedge f(0) > f(1)$

При этом для спецификаций, по которым невозможно синтезировать функцию, можно сразу доказать их нереализуемость путем подстановки конкретных значений для входных параметров функции, как в примере на слайде.



Поиск предусловий ветки

Задача мультиабдукции (вывод неизвестного предиката R_i):

$$\bigwedge_{i=1} \bigwedge_{j=1} R_i(x_{ij}) \implies C$$

Пример мультиабдукции

$$R(x) \wedge R(y) \implies x + y \geq x - y$$

Результат: $R(x) = x \geq 0$

Синтез программ

└ Поиск веток синтезируемой функции

Поиск веток синтезируемой функции



Поиск предусловий ветки

Задача мультиабдукции (вывод неизвестного предиката R_i):

$$\bigwedge_{i=1} \bigwedge_{j=1} R_i(x_{ij}) \implies C$$

Пример мультиабдукции

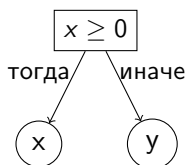
$$R(x) \wedge R(y) \implies x + y \geq x - y$$

Результат: $R(x) = x \geq 0$

Так как реализация функции ищется в виде терма с оператором ветвления, то реализацию можно представить из веток. Ветка — это пара: некоторое значение функции, представленная линейный термом, и предусловие (предикат), при котором синтезируемая функция определяется этим значением (термом).

Для поиска предусловий нам пришла идея использовать мультиабдукцию. Мультиабдукция — это логическая процедура, предложенная в 2016 году.

Эта процедура вывода (синтеза) неизвестных предикатов описана в отчёте.



Поиск линейных термов ветки

Шаблон с неизвестными коэффициентами \bar{c} и k :

$$c_1x_1 + c_2x_2 + \dots + c_nx_n + k$$

Для конкретных значений \bar{x} : $x_1 \mapsto \alpha_1, \dots, x_n \mapsto \alpha_n$:

$$c_1\alpha_1 + c_2\alpha_2 + \dots + c_n\alpha_n + k$$

Синтез программ

2020-06-09

└ Поиск веток синтезируемой функции

Поиск веток синтезируемой функции



Поиск линейных термов ветки
Шаблон с неизвестными коэффициентами \bar{c} и k :
$$c_1x_1 + c_2x_2 + \dots + c_nx_n + k$$

Для конкретных значений \bar{x} : $x_1 \mapsto \alpha_1, \dots, x_n \mapsto \alpha_n$:
$$c_1\alpha_1 + c_2\alpha_2 + \dots + c_n\alpha_n + k$$

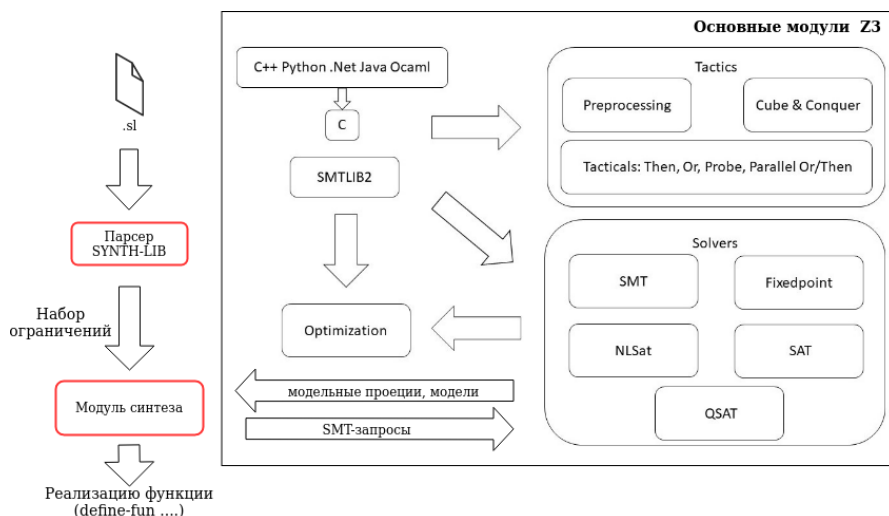
Для поиска значений функции используется шаблон с неизвестными коэффициентами.

Конкретные значения коэффициентов ищутся при некоторых конкретных значениях входных параметров с помощью разрешающей процедуры для фрагмента арифметики.

Алгоритм синтеза по спецификациям с множественными вызовами

- 15 **Листинг 3.** Алгоритм системы по спецификациям с
- 16 **многозначными функциями, использующий одну ветку**
- 17 **вычисления**
- 18 **Вход:** $\{f\} \forall x, y \in \mathcal{D}$
- 19 **функция** $\mathcal{D} \rightarrow \mathcal{D}$ (\mathcal{D} — непустое множество), терм реализации для $f(x)$
- 20 **begin**
- 21 $branches \leftarrow \emptyset;$
- 22 $mlp \leftarrow \emptyset;$
- 23 $i \leftarrow 0;$
- 24 **while** $f(x)$ не определено;
- 25 **do** $f(x) \in \text{opnd}(f, x)$
- 26 $v_i \leftarrow \sqrt[2]{f(x)} / |f(x)|;$
- 27 $i \leftarrow i + 1;$
- 28 **if** $i = 0$;
- 29 $M_{\text{map}} \leftarrow \{v_i, y, f(x) \wedge \bigwedge_{j \in \mathcal{D}} \neg (y = f(j))\};$
- 30 $mlp \leftarrow G(M_{\text{map}}, f(x), M_{\text{map}} \setminus \{y, y\});$
- 31 $mlp \leftarrow \text{mlp} \cup \{v_i, y, f(x) \wedge \bigwedge_{j \in \mathcal{D}} \neg (y = f(j))\};$
- 32 **пока** $M_{\text{map}} \neq \bigcup_{i=1}^n \{v_i, y, f(x) \wedge \bigwedge_{j \in \mathcal{D}} \neg (y = f(j))\} \wedge mlp$
- 33 $\neq \emptyset$;
- 34 $mlp \leftarrow \bigcup_{i=1}^n \{v_i, y, f(x) \wedge \bigwedge_{j \in \mathcal{D}} \neg (y = f(j))\};$
- 35 **если** $i = 0$;
- 36 **то** **вернуть** ($\text{преобразование}, \emptyset$);
- 37 **иначе**
- 38 **для** $v_i \in \text{opnd}(f, x)$;
- 39 **если** $v_i \in \text{prende}(v_i, f(x))$;
- 40 $\{v_i \leftarrow \sqrt[2]{\text{branchend}(f(x), f(x))} / |f(x)|\};$
- 41 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 42 **если** $M_{\text{map}}(v_i, M_{\text{map}}) \neq \emptyset$;
- 43 $v_i \leftarrow \sum_{j \in \mathcal{D}} \{v_i, M_{\text{map}}(f(v_i))\} \cup M_{\text{map}}(f(v_i));$
- 44 $v_i \leftarrow \text{Abduct}(v_i, v_i) \wedge \neg (x = 0);$
- 45 $\{v_i \leftarrow \sqrt[2]{f(x)} / |f(x)| - \sum_{j \in \mathcal{D}} \{v_i, y, f(x) \wedge \bigwedge_{j \in \mathcal{D}} \neg (y = f(j))\}\};$
- 46 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 47 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 48 **иначе**
- 49 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 50 **если** $M_{\text{map}}(v_i, M_{\text{map}}) \neq \emptyset$;
- 51 $v_i \leftarrow \text{Abduct}(v_i, v_i) \wedge \neg (x = 0);$
- 52 $\{v_i \leftarrow \sqrt[2]{f(x)} / |f(x)| - \sum_{j \in \mathcal{D}} \{v_i, y, f(x) \wedge \bigwedge_{j \in \mathcal{D}} \neg (y = f(j))\}\};$
- 53 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 54 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 55 **иначе**
- 56 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 57 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 58 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 59 **иначе**
- 60 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 61 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 62 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 63 **иначе**
- 64 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 65 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 66 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 67 **иначе**
- 68 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 69 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 70 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 71 **иначе**
- 72 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 73 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 74 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 75 **иначе**
- 76 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 77 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 78 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 79 **иначе**
- 80 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 81 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 82 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 83 **иначе**
- 84 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 85 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 86 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 87 **иначе**
- 88 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 89 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 90 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 91 **иначе**
- 92 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 93 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 94 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 95 **иначе**
- 96 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 97 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 98 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 99 **иначе**
- 100 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 101 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 102 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 103 **иначе**
- 104 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 105 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 106 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 107 **иначе**
- 108 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 109 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 110 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 111 **иначе**
- 112 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 113 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 114 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 115 **иначе**
- 116 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 117 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 118 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 119 **иначе**
- 120 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 121 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 122 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 123 **иначе**
- 124 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 125 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 126 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 127 **иначе**
- 128 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 129 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 130 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 131 **иначе**
- 132 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 133 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};$
- 134 $branches \leftarrow \text{branchend} \cup \{v_i\};$
- 135 **иначе**
- 136 $\{v_i \leftarrow \sqrt[2]{\text{branch}(f(x), v_i)} \wedge f(x) / |f(x)|\};$
- 137 $\{v_i \leftarrow \text{prende}(v_i, f(x))\};</$

- C++ – как расширяющий модуль SMT-решателя Z3

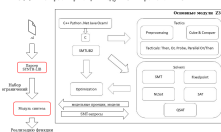


Синтез программ

Реализация MI-SYNT

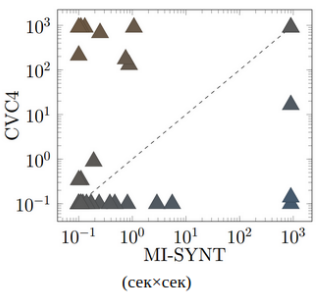
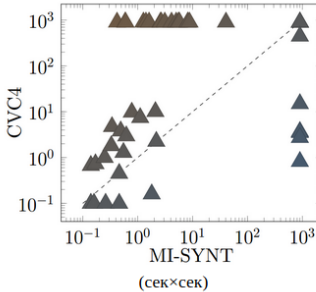
Реализация MI-SYNT

• C++ – как расширяющий модуль SMT-решателя Z3



Эти алгоритмы были реализованы как расширяющий модуль популярного SMT-решателя Z3 на языке C++. Z3 используется для нахождения моделей и модельных проекции.

На вход инструменту подаётся задача синтеза в Synth-Lib формате, который описан в отчете, а на выходе инструмента реализация функции.

Синтезатор	Спецификации с множественными вызовами (87)			
	Нереализуемые спецификации (41)		Реализуемые спецификации (46)	
	Решил	Быстрее	Решил	Быстрее
MI-SYNT	34	8	35	26
CVC4	33	5	24	7
Таймауты размещены на границах				

2020-06-09

Синтез программ

Эксперименты

Синтезатор	Спецификации с множественными вызовами (87)			
	Нереализуемые спецификации (41)		Реализуемые спецификации (46)	
	Решил	Быстрее	Решил	Быстрее
MI-SYNT	34	8	35	26
CVC4	33	5	24	7
Таймауты размещены на границах				

Работоспособность была проверена на 87 спецификациях с множественными вызовами (из них 41 нереализуемых и 46 реализуемых) и 50 спецификациях с одиночным вызовом, взятых с соревнования синтезаторов SyGuS-Comp. Производительность была сравнена с известным синтезатором CVC4. На слайде представлены только спецификации с множественными вызовами, так как задачи с одиночными вызовами решили оба инструмента. Реализованным инструментом MI-SYNT решил в течение 10 минут 69 спецификаций с множественными вызовами, в то время как CVC4 смог решить только 55 задач. Также MI-SYNT опередил по времени исполнения CVC4 на 23 задачах с множественными вызовами.

- Описаны ключевые понятия функционального синтеза.
- Разработан алгоритм синтеза функций по декларативным спецификациям теории линейной арифметики с одиночным вызовом
- Разработан алгоритм синтеза функций по спецификациям теории линейной арифметики с множественными вызовами
- Был реализован программный инструмент MI-SYNT как расширяющий модуль синтеза по спецификациям с множественными вызовами для решателя Z3 на языке C++
- Проведено экспериментальное исследование на 87 спецификациях с множественными вызовами и 50 спецификациях с одиночным вызовом

Синтез программ

2020-06-09

Результаты

Результаты

- Описаны ключевые понятия функционального синтеза.
- Разработан алгоритм синтеза функций по декларативным спецификациям теории линейной арифметики с одиночным вызовом
- Разработан алгоритм синтеза функций по спецификациям теории линейной арифметики с множественными вызовами
- Был реализован программный инструмент MI-SYNT как расширяющий модуль синтеза по спецификациям с множественными вызовами для решателя Z3 на языке C++
- Проведено экспериментальное исследование на 87 спецификациях с множественными вызовами и 50 спецификациях с одиночным вызовом

- Описаны ключевые понятия функционального синтеза.
- Разработан алгоритм синтеза функций по декларативным спецификациям теории линейной арифметики с одиночным вызовом с использованием модельных проекций и подстановки вместо вызовов функции шаблонов для термов.
- Разработан алгоритм синтеза нерекурсивных функций по спецификациям с множественными вызовами, использующий мультиабдукцию для поиска предусловий.
- Реализован программный инструмент MI-SYNT как расширяющий модуль синтеза по спецификациям с одиночными и множественными вызовами для решателя Z3 на языке программирования C++.
- Проведено экспериментальное исследование на 137 спецификациях. MI-SYNT смог решить 69 спецификаций с множественными вызовами, а CVC4 решил только 55 задач.