Санкт-Петербургский государственный университет

Кафедра системного программирования

Мишенев Вадим Сергеевич

Синтез программ по спецификациям с множественными вызовами

Дипломная работа

Научный руководитель:
д. т. н., профессор Кознов Д. В.

Научный консультант:
доцент Федюкович Г. Г.

Рецензент:
ООО «ИнтеллиДжей Лабс»
разработчик ПО Косарев Д. С.

Санкт-Петербург
2020

SAINT-PETERSBURG STATE UNIVERSITY

Software Engineering

Mishenev Vadim

Program synthesis on multi-invocation specifications

Graduation Thesis

Scientific supervisor:
Doctor of Engineering, Professor D. V. Koznov

Consultant:
Ph.D., Assistant Professor G. G. Fedyukovich

Reviewer:
"IntelliJ Labs Co.Ltd"
Software Developer D. S. Kosarev

Saint-Petersburg
2020

Оглавление

Введение

Широко известна задача автоматического создания программ на некотором языке программирования, которые удовлетворяют намерениям пользователя, выраженным в форме ограничений [7, 10, 21]. Она называется задачей синтеза программ. Качественный синтез программ делает процесс разработки более надежным и менее трудоёмким. Она лежит на стыке следующих областей — искусственного интеллекта, машинного обучения, формальной верификации, математической логики.

Среди известных подходов к синтезу программ существует эффективный и хорошо масштабируемый подход, называемый функциональным синтезом [8, 9, 15, 16, 17, 18, 19, 27, 30, 35]. В рамках этого подхода синтезируемая программа рассматривается как функция, а намерения пользователя выражаются в виде утверждений на языке исчисления предикатов первого порядка, называемых спецификацией программы, которая связывает входные и выходные параметры функции.

Функциональный синтез накладывает сильные ограничения на формат спецификации. Существующие методы функционального синтеза основываются на эффективных алгоритмах устранения кванторов [8, 15, 16, 17, 19, 27, 30, 35]. Однако в настоящий момент они хорошо работают только для спецификаций, в которых синтезируемая функция встречается только с одним и тем же фиксированным набором аргументов (так называемые спецификации с одиночным вызовом)[1]. Таким образом, производительность синтезаторов, а также успешность генерации программы в решающей степени зависит от того, как сформулированы спецификации.

В действительности часто оказывается, что строго формулировать уточнения, которые бы адекватно представляли целевые намерения, представляющие собой спецификацию, очень затратно. Это, во многом, объясняется тем, что спецификации либо генерируются другими автоматическими инструментами, либо описываются программистами

[1]Альтернативное определение: спецификаций, имеющих вхождения синтезируемой функции только с одним и тем же фиксированным набором аргументов. Например, спецификация, имеющая два вхождения функции с разными аргументами: $f(x) > f(x-1)$.

вручную. Во многих случаях может даже оказаться, что невозможно сформулировать дополнительные уточняющие ограничения на целевую функцию. В то же время спецификации с множественными вызовами, где функция может входить в неё с различными наборами аргументов, значительно более выразительны, чем с одиночными, однако и проблема синтеза функций по ним значительно сложнее. Частным случаем таких спецификаций является спецификация с одиночным вызовом. Возможность синтеза программ по спецификациям с множественными вызовами расширит область применимости синтеза и упростит процесс описания ограничений. Таким образом, адаптация функционального синтеза к спецификациям с множественными вызовами является открытой и актуальной проблемой.

Рассматривая только спецификации в линейной арифметике, искомую функцию удобно представлять в виде набора пар: линейного выражения, представляющего собой выходное значение функции при заданных входных параметрах, и условия, являющегося предикатом относительно входных параметров, при котором справедливо это выражение. Каждая такая пара представляет собой ветку функции. Таким образом, для решения этой проблемы предполагается искать ветки, используя модели спецификации и сведение к задаче поиска неизвестных предикатов, называемой мультиабдукцией [1]. При этом для каждой модели проверять существование функции для заданных ограничений, т.е. реализуемость спецификации, с помощью решателя для теории неинтерпретированных функций. Кроме того, для упрощения исходной спецификации и эффективного поиска веток предполагается использование модельных проекций [6, 25].

1. Постановка задачи

Целью данной работы является создание алгоритма синтеза программ по логическим спецификациям с множественными вызовами. Для достижения поставленной цели были сформулированы следующие 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.

2. Обзор

2.1. Синтез программ

Синтез программ можно рассматривать в трех ключевых плоскостях: виды ограничений, которые выражают пользовательские намерения, пространство поиска программ и процедура поиска в этом пространстве.

Пользовательские намерения могут быть выражены в различных формах, включая логические спецификации, примеры исполнения программы, трассы, естественный язык, частичные программы или даже связанные программы для их супероптимизации.

Пространство поиска программ обычно ограничено синтаксически, например, подмножеством некоторого языка программирования. Ограничения также могут быть выражены в виде наброска программы, формальной грамматики или предметно-ориентированного языка. Процедура поиска желаемой программы может быть основана на перечислительном поиске, индуктивном, дедуктивном синтезе, статистических методах или их комбинации. Инструмент, осуществляющий поиск реализации желаемой программы, называют синтезатором.

Синтез программ имеет множество применений: автоматическое исправление ошибок в исходном коде, обработка данных (FlashFill — технология автозаполнения данных в Excel [20]), автодополнение кода, супероптимизация, т. е. поиска эквивалентной в семантическом смысле программы, содержащей оптимальную последовательность инструкций, автоматического поиска размещения минимального количества синхронизаций в параллельной программе и другое.

Область приложения во многом зависит от того, как сформулированы ограничения и какое пространство программ-кандидатов.

2.2. Основные определения

Сигнатура многосортной теории первого порядка состоит из трех непересекающихся множеств $\mathcal{S}, \mathcal{F}, \mathcal{P}$, соответственно, множеств сортов,

функциональных и предикатных символов. В отличие от обычной теории первого порядка аргументы функциональных и предикатных символов, а также сами функции имеют сорт. Тогда формулы и термы теории определяются рекурсивно следующим образом:

$$\begin{aligned} \mathit{trm} &::= f(\mathit{trm}, \ldots, \mathit{trm}) \mid \mathit{ite}(\mathit{fla}, \mathit{trm}, \mathit{trm}) \mid \mathit{const} \mid \mathit{var} \\ \mathit{fla} &::= \top \mid \bot \mid p(\mathit{trm}, \ldots, \mathit{trm}) \mid \mathit{Bvar} \mid \neg \mathit{fla} \mid \mathit{fla} \wedge \mathit{fla} \mid \mathit{fla} \vee \mathit{fla} \end{aligned}$$

Здесь $f \in \mathcal{F}$, терм *ite* соответствуют если-тогда-иначе оператору, терм *const* является функциональным символом $v \in \mathcal{F}$ арности 0, *var* (*Bvar*) является переменной сорта $\mathcal{S}$ (соответственно, *Bool*), константы $\top$ и $\bot$ имеют сорт *Bool*, и $p \in \mathcal{P}$.

В частности, для линейной целочисленной арифметики (LIA), $\mathcal{S} \stackrel{\text{def}}{=} \{\mathbb{Z}, \mathit{Bool}\}$, $\mathcal{F} \stackrel{\text{def}}{=} \{+, \cdot\}$ (и применения $\cdot$ одновременное к двум и более переменным запрещено), и $\mathcal{P} \stackrel{\text{def}}{=} \{>, <, \geq, \leq, =\}$.

Тогда структура M состоит из непустого и попарно непересекающихся доменов (множеств) M_s для каждого сорта $s \in \mathcal{S}$ и интерпретирует функциональные $f \in \mathcal{F}$ и предикатные $p \in \mathcal{F}$ символы отображениями M_f и M_p соответственно их арности и сортам.

Оценкой v называется тотальное отображение переменных в соответствующие носители (домены). Истинностное значение формулы φ в интерпретации M при некоторой оценке v определяется как обычно [44]. При этом оценка не влияет на истинность замкнутых формул.

Структура M называется моделью φ, если формула φ истинна в этой структуре при любых оценках v, и обозначается как $M \models \varphi$.

Формула φ называется выполнимой, если у нее есть модель, т.е. существует M такая, что $M \models \varphi$. Иначе формула φ невыполнима.

Формула φ общезначима (записывается как $\models \varphi$), если $M \models \varphi$ для всех структур M.

Формула A называется логическим следствием множества формул Γ (обозначается как $\Gamma \models A$), если для всякой интерпретации M и любой оценки v выполняется: если для любой формулы $G \in \Gamma$ верно $M, v \models G$, то $M, v \models A$. Для обозначения подстановки формулы u вместо терма t

в формуле φ пишется $\varphi[u/t]$.

2.3. Постановка задачи синтеза

В последнее время пользовательские намерения выражаются в виде логической формулы, так как это достаточно универсальный способ представить ограничения на синтезируемую программу. Такую логическую формулу называют спецификацией. Спецификация связывает входные и выходные параметры функции. Таким образом, рассматривается синтез некоторой вычислимой функции автоматически от некоторой логической формулы, описывающей свойства функции.

Спецификация определяет проблему функционального синтеза и в общем случае имеет следующий вид:

$$\exists f. \forall \bar{x}. \varphi(f, \bar{x}). \tag{1}$$

Здесь φ является формулой некоторой теории первого порядка[2], $\bar{x}$ — это вектор переменных, от которых зависит синтезируемая функция, а f — функциональный символ. При этом мы требуем тотальность для функции f, т.е. функция должна быть определена для любых наборов входных параметров $\bar{x}$.

Спецификация рассматривается как формула сигнатуры некоторой формальной теории T, в которой все переменные универсально квантифицированы. Так как в общем случае разные теории требуют различные методы синтеза, то рассматривается только теория линейной целочисленной арифметики.

Задача функционального синтеза состоит в нахождении терма f_{impl} теории T такого, что выражение $\varphi[f/f_{impl}]$, полученное заменой в φ из спецификации всех вхождений f на f_{impl}, станет выполнимой по модулю теории T. При этом f_{impl} рассматривается как реализация функции f. Множество всевозможных термов определяет пространство поиска функции. Если такого терма f_{impl} не существует, то соответствующая спецификация называется нереализуемой.

[2]Для краткости под φ будет обозначаться спецификация.

Так как рассматривается теория линейной целочисленной арифметики с сигнатурой, определенной выше, то f_{impl} будет состоять из возможно вложенных друг в друга *ite*-операторов и линейных термов. Также это удобно представлять в виде решающего дерева, в листьях которого — линейные термы, а во внутренних узлах — формулы. Кроме того, функцию можно представлять в виде набора пар: линейного выражения, представляющего собой выходное значение функции при заданных входных параметрах, и условия, являющегося формулой относительно входных параметров, при котором справедливо это выражение. Каждая такая пара представляет собой ветку функции.

Спецификации можно разбить на два класса: одиночного вызова (single-invocation) и множественного вызова (multi-invocation) [23]. Свойством одиночного вызова обладает любая формула вида $Q[x, f(x)]$, полученная заменой y на $f(x)$ в формуле $Q[x, y]$, не содержащей кванторов и f. Иначе говоря, функция f входит всюду в формулу Q с одним и тем же набором аргументов. В противном случае рассматриваемая формула обладает свойством множественного вызова. Спецификации с множественными вызовами содержат вхождения f с более чем одним набором аргументов, поэтому они могут намного выразительнее спецификаций с одиночным вызовом, поскольку спецификация с одиночным вызовом может рассматриваться как частный случай спецификации с множественными вызовами. Проблема синтеза по таким спецификациям неразрешима [42] и сводится, например, к проблеме синтеза индуктивных инвариантов [34].

Приведём в качестве примера общий вид спецификации с двумя вызовами:
$\exists f \forall \bar{x}. \forall \bar{y}. \Phi[\bar{x}, \bar{y}, f(\bar{x}), f(\bar{y})]$.

Более частный примером является спецификация: $\exists f \forall x. f(x+1) > f(x)$.

Функциональный синтез находит применения в синтезе функциональных программ [35, 8], синтезе функций внешних вызовов для символьного исполнения [4, 40]. В частности, существует Sketch-фреймворк [26], который позволяет программисту писать не весь код программы, а

оставлять в нём «дырки», которые в дальнейшем будут синтезированы фреймворком в соответствии с ограничениями. Кроме того, такой синтез находит применения в реактивных системах [14, 41], например, при синтезе выигрышных стратегий в играх, которые используются при планировании движения роботов.

Так как эта задача синтеза достаточно популярна, существует соревнование SyGuS-Comp среди синтезаторов программ [36].

2.4. Синтаксически-управляемый синтез

Синтаксически-управляемый синтез (Syntax-Guided Synthesis) – задача синтеза, в которой формализуется проблема синтеза программ не только логической спецификацией, но и дополняется предоставленным пользователем синтаксическим шаблоном для ограничения пространства возможных программ [39]. В такой задаче синтеза синтаксические ограничения выражены в виде множества выражений E, заданных в виде формальной грамматики

Тогда задача синтаксически-управляемого синтеза заключается в поиске выражения $f_{impl} \in E$. Формальная грамматика ограничивает синтаксически терм, рассматриваемый как f_{impl}, некоторой теории T.

Основное достоинство данного синтеза — возможность его применить к спецификациям с множественными вызовами, однако данный подход плохо масштабируется.

2.5. Перечислительный синтез

Большинство решателей SyGuS работают в два этапа: этап обучения, на котором предлагается программа кандидата P, и этап верификации, на котором предложение P проверяется на соответствие спецификации.

Этап верификации осуществляется с помощью оракула, так называемый оракулом-управляемый индуктивный синтез (oracle-guided inductive synthesis), частным случаем которого является синтез, управляемый контрпримерами (counterexample-guided inductive synthesis) [2, 13, 22,

24, 33]. При последней схеме синтезатора оракул возвращает не только да или нет (удовлетворяет ли кандидат спецификации), но и ещё и контрпример, т.е. вход для P, который не удовлетворяет спецификации φ.

На этапе обучения применяются перечислительные методы. Ключевая идея перечислительных методов является структуризация пространства поиска программ с использованием некоторых метрик, таких как размер программы, сложность и т. д., а затем перебор программ в пространстве кандидатов с его сокращением для эффективного поиска программы, которая удовлетворяет спецификации.

2.5.1. Наивные подходы

Наивные перечислительные алгоритмы для синтаксически-управляемого синтеза осуществляют простой перебор всевозможных выражений грамматики. Среди них можно выделить два подхода: сверху-вниз и снизу-вверх.

В алгоритме сверху-вниз перебор выражений начинается со стартового нетерминала грамматики. При этом поддерживается упорядоченный список частичных кандидатов-программ. Для каждого кандидата все вхождения нетерминалов заменяются на всевозможные результаты применения продукций грамматики.

В алгоритме снизу-вверх перебор программ начинается с листьев, т.е. терминальных символов. При этом пропускаются выражения, которые эквивалентны в семантическом смысле уже рассмотренным программам.

Очевидно, что такие алгоритмы не являются эффективными, так как требуют перебор экспоненциального числа всевозможных выражений грамматик, что особенно актуально для синтезируемых программ большего размера.

2.5.2. Синтез программ методом Разделяй и Властвуй

Другой подход — масштабируемый перечислительный синтез программ методом Разделяй и Властвуй [3], реализованный в EuSolver — победителе SyGuS-Comp 2016 [37]. Он применим для спецификаций, содержащих ровно одну неизвестную синтезируемую функцию, которая является поточечно-заданной (связывает вход с выходом, но не наоборот), т.е. в синтаксическом смысле означает, что все вхождения синтезируемой функции имееют одни и те же аргументы. Ключевая идея подхода состоит в том, чтобы исходную формальную грамматику разделить на две: грамматику для предикатов и грамматику для термов, поскольку условные выражения состоят непосредственно из самого условия — предиката и возвращаемого значения — терма. При таком разделении синтезируемой программе можно поставить в соответствие дерево решений, во внутренних узлах которого — предикаты, а в листьях — термы.

Дерево строится из сгенерерированных термов и предикатов на основе критерия информативности — энтропии. Отсюда следует недостаток этого метода — перебор термов и предикатов осуществляется независимо, при этом никак не задействована их семантика.

2.6. Synth-Lib формат

SyGuS-задача описывается во входном формате Synth-Lib, основанном на языке SMT-Lib для SMT-решателей [29]. Этот формат подходит и для описания задач без синтаксических ограничений.

Ниже приведён пример спецификации программы для поиска максимума из двух чисел в теории линейной арифметики:

$$\exists max2 \forall x \forall y . max2(x,y) \geq x \wedge max2(x,y) \geq y \\ \wedge (max2(x,y) = x \vee max2(x,y) = y). \quad (2)$$

Задача синтеза функции максимума двух чисел в формате Synth-Lib будет иметь вид:

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int)
(declare-var x Int)
(declare-var y Int)
(constraint (>= (max2 x y) x))
(constraint (>= (max2 x y) y))
(constraint (or (= x (max2 x y)) (= y (max2 x y))))
(check-synth)
```

В первой строке `(set-logic LIA)` устанавливается теория целочисленной линейной арифметики, которой принадлежит рассматриваемая спецификация. Во второй строке `(synth-fun max2 ...)` описывается синтезируемая функция — имя, типы параметров, возвращаемый тип, опционально (в зависимости от версии формата, в более ранних — обязательно) синтаксические ограничения.

Семантические ограничения задаются в строках вида `(constraint ...)`. Строки вида `(declare-var ...)` определяют универсально квантивицированные переменные и их типы.

Строка `(check-synth)` непосредственно инициирует синтез. В результате будут сгенерированы только те функции, которые определены до этой команды. Аналогично и для ограничений.

Дополнительно с помощью команды `define-fun` можно определять макросы.

В зависимости от версии Synth-Lib формата меняются наборы команды и предъявляемые требования к ним. Например, из второй версии языка удалена команда `declare-fun` для неинтерпретируемых функций [28].

Решением такой спецификации может быть функции в языке линейный термов с оператором если-тогда-иначе: $max2(x, y) = ite(x < y, y, x)$.

2.7. Функциональный синтез

В отличие от синтаксически-управляемого синтеза синтаксические ограничения игнорируются, поэтому тело синтезируемой функции f

представляется синтаксически в виде терма.

2.7.1. Синтез, основанный на опровержении

Одна из последних идей разрешения проблемы синтеза — это подход, основанный на опровержении [9, 30]. Данный подход реализован в CVC4, который является победителем SyGuS-Comp 2017, а также SyGuS-Comp 2018 (в четырёх треках из пяти) [38].

Рассматриваемый подход получает решение из доказательства невыполнимости отрицания исходной спецификации. В статье [30] рассматриваются задачи синтеза по спецификациям с одиночным вызовом. Тогда для спецификации вида $\exists f \forall \bar{x} Q[\bar{x}, f(\bar{x})]$ существует логически эквивалентная ей формула: $\forall \bar{x} \exists y Q[\bar{x}, y]$.

Данный подход предполагает установление невыполнимости отрицания предыдущей формулы:

$$\exists \bar{x} \forall y \neg Q[\bar{x}, y]. \tag{3}$$

По этой формуле можно построить сколемовскую версию (по аксиоме выбора) $\forall y \neg Q[\bar{\mathbf{k}}, y]$, где $\bar{\mathbf{k}}$ — вектор сколемовских констант (новых нульместных функциональных символов из счетного множества $Scol$, $Scol \cap \mathcal{F} = \varnothing$).

При такой постановке задачи существует решение для $f(\bar{x})$ следующего вида:

$$ite(Q[\bar{x}, t_p], t_p, (...ite(Q[\bar{x}, t_2], t_2, t_1)...)). \tag{4}$$

При этом $t_1[\bar{x}], ..., t_p[\bar{x}]$ — это термы для множества формул
$\Gamma = \{\neg Q[\bar{x}, t_1[\bar{x}]], ..., \neg Q[\bar{x}, t_p[\bar{x}]]\}$, которые, в свою очередь, невыполнимы относительно теории T.

Однако выбор такого терма $t[\bar{x}]$, в частности, когда он зависит от $\bar{x}$, является эвристическим. Успех процедуры синтеза на практике во многом зависит от этой эвристики.

Таким образом, можно найти реализацию синтезируемой функции $f(x)$ путем опровержения формулы (3) (инстанцирование квантора, управляемого контр-примерами).

Рассмотренной метод может применяться и для SyGuS-задачи. Тогда утверждения и ограничения такой задачи, в том числе и синтаксические, можно закодировать в некоторую теорию T_{ev} — расширение исходной теории T. Основная идея состоит в том, чтобы представить исходную грамматику как набор (алгебраических) типов данных, интерпретация которых в терминах исходной теории T передается решателю.

2.7.2. Синтез методом ленивой элиминации квантора

Этот подход применим к синтезу нерекурсивных функций по спецификациям с одиночными вызовами. В основе такого подхода лежит ленивая элиминация квантора с использованием модельных проекций [16]. В то время как при наивном подходе требуется затратное преобразование спецификации к дизъюнктивно нормальной форме и затем применение квантовой элиминации (получение эквивалентной бесквантворной формулы [31]) к каждому дизъюнкту [8, 35].

Такой подход позволяет быстрее находить решения для задач функционального синтеза. В частности, он намного эффективнее синтаксически-управляемого синтеза для больших и выразительных спецификациий, однако он применим только для спецификаций с одиночным вызовом.

При таком подходе в процессе элиминации квантора существования строится дерево решений, которое представляет собой реализацию искомой (сколемовской) функции. При этом сама исходная задача синтеза разбивается на подзадачи с помощью функциональной декомпозиции, где каждая подзадача представляется в терминах предусловий (precondition) и сколемовских ограничений. Затем по сколемовским ограничениям находится сколемовский линейный терм в рамках теории линейной арифметики, представляющий вместе с соответствующим предусловием некоторую ветку функции.

Как и в синтезе, основанном на опровержении, исходную спецификацию вида $\exists f \forall \bar{x} Q[\bar{x}, f(\bar{x})]$ можно представить: $\forall \bar{x} \exists \bar{y}.Q[\bar{x}, \bar{y}]$. В данном методе в спецификации одновременно может присутствовать больше одной синтезируемой функции с одиночным вызовом, поэтому в общем

виде $\bar{y}$ — вектор выходных значений одной общей функций.

Определение 2.1. Декомпозицией выполнимой формулы $\exists \bar{y}.\psi(\bar{x}, \bar{y})$ является пара $(pre; \phi)$, где pre (называемые предусловиями) — это вектор формул длины M, ϕ (называемые сколемовскими ограничениями) — это матрица размерности $M \times |\bar{y}|$ такая, что выполняются утверждения, представленные ниже.

$$\models \bigvee_{i=1}^{M} pre[i](\bar{x}) \tag{5}$$

$$pre[i](\bar{x}) \wedge \bigwedge_{j=1}^{|y|} \phi[i,j](\bar{x}, \bar{y}) \models \psi(\bar{x}, \bar{y}) \tag{6}$$

$$pre[i](\bar{x}) \models \exists \bar{y} \bigwedge_{j=1}^{|y|} \phi[i,j](\bar{x}, \bar{y}) \tag{7}$$

Лемма 2.1. *Для любой формулы вида $\exists \bar{y}.\psi(\bar{x}, \bar{y})$ существует функциональная декомпозиция.*

Для любой формулы можно в качестве предусловий взять $true$, а в качестве ограничений — саму формулу.

Теорема 2.2. *Пусть $(pre; \phi)$ — декомпозиция формулы $\exists \bar{y}.\psi(\bar{x}, \bar{y})$ и sk — матрица сколемовских термов. Тогда Sk_j - сколемовский терм для $\bar{y}[j]$:*

$$Sk_j = ite(pre[1], sk[1; j], ...ite(pre[M-1], sk[M-1; j], sk[M; j])) \tag{8}$$

Сколемовские линейные термы Sk можно получить из сколемовских ограничений ϕ с помощью процедуры, описанной в статье [16].

Для получения предусловий и соответствующих ограничений процедура синтеза используют последовательность модельных проекций (MBP — Model-Based Projections) [6, 25].

Определение 2.2. $MBP_{\bar{y}}$ — отображение с конечным числом образов из моделей формулы $\psi(\bar{x}, \bar{y})$ в бескванторную формулу над свободными переменными $\bar{y}$. При этом должны выполняться условия:

Листинг 1: Основной алгоритм синтеза спецификаций с одиночными вызовами AEVAL($\exists y.\psi(\bar{x},\bar{y})$)

Вход: $\exists y.\psi(\bar{x},\bar{y})$
Выход: Реализуемость, предусловия pre, сколемовские ограничения ϕ

```
1 M ← 1.
2 пока true
3 |   если ⊭ ⋀_{i=1}^{M} ¬pre[i](x̄)  вернуть (РЕАЛИЗУЕМА, pre, φ) ;
4 |   tmp ← φ(x̄, ȳ) ∧ ⋀_{i=1}^{M} ¬pre[i](x̄).
5 |   если ∃m m ⊨ tmp
6 |   |   (pre[M]; φ[M; 1]; ...φ[M; |ȳ|]) ← GetMBP(ȳ; m; ψ).
7 |   |   M ← M + 1.
8 |   иначе
9 |   |   вернуть (НЕРЕАЛИЗУЕМА, 0, 0)
```

1) если $m \models \psi(\bar{x},\bar{y})$, то $m \models MBP_{\bar{y}}(m,\psi)$;
2) $MBP_{\bar{y}}(m,\psi) \models \exists\bar{x}.\psi(\bar{x},\bar{y})$ и $\exists\bar{x}.\psi(\bar{x},\bar{y}) \equiv \vee_i MBP_{\bar{y}}(m_i,\psi)$.

Другими словами, модельные проекции покрывают все множество моделей $\psi(\bar{x},\bar{y})$ конечным множеством бесквантворных формул над $\bar{y}$. Модельные проекции существуют для теорий, допускающих кванторную элиминацию, в том числе для теории линейной арифметики.

Основной алгоритм представлен в листинге 1. Главный цикл программы продолжается до тех пор, пока не будет выполнено первое свойство функциональной декомпозиции, иначе говоря, пока не будут найдены все предусловия. На строке 4 получаем некоторый контрпример, т.е. модель, которая покрывается не всеми модельными проекциями или предусловиями. По данному контр-примеру уточняется дерево решений (строки 6-7) с помощью модельной проекции, полученной процедурой $GetMBP$.

Очевидно, что от числа итераций M зависит напрямую глубина решающего дерево, которому соответствует сколемовская функция. В статье [16] описана процедура, уменьшающая высоту решающего дерева в случае, когда генерируемая функция имеет больше одного выхода

$\bar{y}$. Основная идея такой процедуры — объединение предусловий для веток, имеющих общий выход.

Данный метод синтеза является эффективным и масштабируемым по сравнению с рассмотренными ранее подходами благодаря модельным проекциям.

2.8. Z3-решатель

Методы функционального синтеза активно используют решатели для задачи выполнимости формул в теориях (satisfiability-modulo theory, SMT) [11, 32]. Такая задача является задачей разрешимости логической формулы некоторой формальной теории. В случае выполнимости логической формулы решатели находят модель, при которой истина эта формула.

Одним из эффективных решателей таких задач является Z3 [43], написанный на C++. Он поддерживает такие теории, как арифметику, теорию массивов, битовых векторов, неинтерпретированных функций и другие.

Кроме того, он позволяют элиминировать кванторы для теории линейной арифметики. Z3 имеет внешний программный интерфейс (api) для разных языком программирования, а также имеет возможность чтения SMT-задач из файла формата .smt (SMT-Lib 2 [5]), для этого в нём реализованы лексический и синтаксический анализаторы.

Логические выражения представляются в Z3 как абстрактное синтаксическое дерево. Кроме того, предоставляется набор инструментов для их преобразования (например, реализация подстановки одного подвыражение на другое, приведение к каноническому виду и т.д.). Общая архитектура системы Z3 представлена на рисунке 1. Само ядро Z3 содержит SMT-решатели для разных теорий, SAT-решатель. При этом каждый решатель имеет свой контекст. Присутствуют важные инструменты для устранения кванторов, модельных проекций, в частности, которые необходимы для алгоритма синтеза.

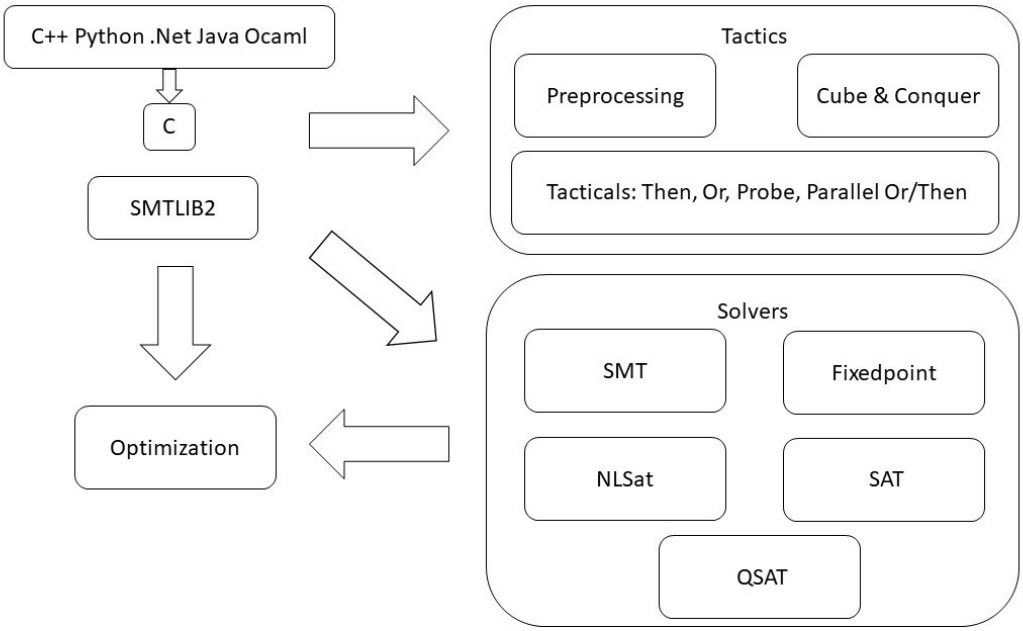

Рис. 1: Общая архитектура системы Z3

3. Синтез по спецификациям с одиночным вызовом

Частным случаем синтеза по спецификациям с множественными вызовами является синтез по спецификациям с одиночным вызовом.

Поиск реализации функции $f(\bar{x})$ можно свести к поиску множества веток, т.е. пар $\langle \pi, \beta \rangle$: выходного линейного терма β и предусловия π, при котором выходное значение функции можно представить этим термом. При этом в силу тотальности $f(\bar{x})$ дизъюнкция всех предусловий $\bigvee^i \pi_i$ должна быть общезначимой.

3.1. Поиск ветвей синтезируемой функции

Одной из главных задач рассмотренных методов является поиск термов синтезируемых веток программы.

Линейный терм каждой ветки синтезируемой программы можно представлять в виде шаблона ветки, т.е. линейной комбинации над $\bar{x}$ вида:

$$\bar{c}[1] * \bar{x}[1] + \bar{c}[2] * \bar{x}[2] + \ldots + \bar{c}[n] * \bar{x}[n] + k. \tag{9}$$

где $\bar{c}$ — вектор новых констант для коэффициентов, k — новая константа. Для поиска конкретных значений коэффициентов $\bar{c}$ и k можно использовать модель M_x формулы φ путем подстановки конкретных значений вместо универсально квантифицированных переменных $\bar{x}$ в (9), тем самым получая следующий терм отдельной ветки β_i для f:

$$\bar{c}[1] * M(\bar{x}[1]) + \bar{c}[2] * M(\bar{x}[2]) + \ldots + \bar{c}[n] * M(\bar{x}[n]) + k. \tag{10}$$

После чего, подставив эту ветку вместо одиночного вызова f, т.е. вместо всех вхождения $f(\bar{x})$, можно получить модель для коэффициентов M_{coeff} и, соответственно, их конкретные значения.

Интуитивно понятно, что обе модели M_x и M_{coeff} представляют некоторое единичное поведение возможной реализации синтезируемой функции f, в которой коэффициенты из M_{coeff} гарантированно вычис-

ляют правильное выходное значение только для конкретных входных параметров, предоставленных M_x.

Гипотетически возможно было продолжать перечислять всевозможные входные значения с помощью различных моделей M_x и строить ветви для них. Предусловия таких ветвей будут просто содержать равенства между входными параметрами функции и соответствующими значениями.

Таким образом, основные цели при поиске ветвей: коэффициенты и константа должны обеспечивать как можно более общее поведение, и, соответственно, более общие предусловия. Последнее актуально для для синтеза по спецификациям с множественными вызовами.

Однако ключевой задачей остается поиск предусловий. После получения конкретного терма ветки β_i можно попытаться найти предусловие, т.е. такую формулу π_i, для которой верно, что $\pi_i(\bar{x}) \implies \varphi[\beta_i(\bar{x})/f(\bar{x})]$, где β_i – ветка вида (9) с конкретными коэффициентами. В этом тривиальном случае, когда посылка импликации содержит только неизвестный предикат и ничем не усилена, $\pi_i = \varphi[\beta_i(\bar{x})/f(\bar{x})]$.

3.2. Использование модельных проекций

Для спецификаций с одиночными вызовом $\forall\bar{x}\exists f.\varphi[f(\bar{x}), \bar{x}]$ можно заменить вызов, т.е. вхождение $f(\bar{x})$, на новую переменную y, тем самым перейти к формуле теории первого порядка вида $\forall\bar{x}\exists y.\varphi[y, \bar{x}]$.

Для поиска предусловий целесообразно использовать, как и в методе ленивой элиминации квантора 2.7.2, модельные проекции по переменной y для формулы φ и её моделей, однако не во всех случаях модельная проекция может представлять собой предусловие для линейного терма ветки.

Для спецификации задачи поиска максимума из двух чисел (2) для любой модели модельная проекция даёт правильные соответствующие предусловия. Например, сначала заменим $max2(x, y)$ на новую перемен-

ную y_{out}, получив следующее выражение:

$$\exists y_{out} \forall x \forall y. y_{out} \geq x \wedge y_{out} \geq y \wedge y_{out} = x \vee y_{out} = y). \quad (11)$$

Сделав запрос к SMT-решателю для φ, получим модель: $M = \{x \mapsto 0; y \mapsto 1; y_{out} \mapsto 1\}$. Затем собираем литералы, которые истины при модели M, в множество: $\{y_{out} \geq x; y_{out} \geq y; y_{out} = y\}$. Выполнив элиминацию квантора для этого множества по переменной y_{out}, получим предусловие для одной ветки: $x < y$. Аналогичным образом при другой модели можно получить предусловие для второй ветки.

Однако модельная проекция не для всех спецификаций выдает корректные предусловия. Например, $\forall x, y. \exists f. f(x, y) \geq x \wedge f(x, y) \geq y.$ для любой модели модельная проекция будет $true$, хотя реализация функции будет $f(x, y) = ite(x \geq y, x, y)$ с двумя ветками $\langle x \geq y, x \rangle$ и $\langle x < y, y \rangle$.

Таким образом, модельные проекции можно эффективно применять для упрощения исходной задачи синтеза, рассматривая не всё пространство входных параметров синтезируемой функции (универсально квантивицированных переменных), а только некоторую его часть, ограниченную модельной проекцией, т.е. искать ветки не для φ, а для $\varphi \wedge \psi$, где φ — бескванторная часть спецификации, а ψ — некоторая модельная проекция. При этом справедливо, что конечная дизъюнкция всех модельных проекций φ будет эквивалентна $\exists y. \varphi$, что в силу тотальности спецификации есть $true$.

3.3. Алгоритм

Псевдокод процедуры для синтеза программ по спецификациям с одиночным вызовом представлен в листинге 2.

Процедура принимает на вход объявление функции $f(\bar{x})$, реализацию которой нужно синтезировать, и спецификацию φ над f. Алгоритм итеративно строит множество ветвей с соответствующими предусловиями, которые покрывают всевозможное значения входных параметров функции f.

Алгоритм завершается, когда найдены все предусловия, покрыва-

Листинг 2: Алгоритм синтеза по спецификациям с одиночным вызовом SiSynt($\exists f.\forall \bar{x}.\varphi(f,\bar{x})$)

Вход: $\exists f.\forall \bar{x}.\varphi(f,\bar{x})$
Выход: $res \in \{$(не)реализуема$\}$, терм реализации для $f(\bar{x})$

```
1  preconds ← ∅;
2  branches ← ∅;
3  пусть y новая переменная;
4  пусть τ̄ аргументы вызова функции f;
5  φ_y ← φ[y/f(τ̄)];
6  пока ⊨ ⋀_{π∈preconds} ¬π(x̄)
7      M_mbp ⊨ φ_y(y, x̄) ∧ ⋀_{π∈preconds} ¬π(x̄)
8      mbp ← GetMBP(y; M_mbp; ∃y.φ_y)
9      пока ∃M_x.M_x ⊨ ⋀_{π∈preconds} ¬π(x̄) ∧ mbp
10         ω ← φ[M_x(x̄)/x̄];
11         если ⊭ ω
12             вернуть ⟨нереализуема, ∅⟩;
13         иначе
14             пусть c̄ и k новые переменные;
15             ω ← φ[ Σ_i (τ̄[i] · c̄[i]) + k/f(τ̄)];
16             если ∃M_coeff . M_coeff ⊨ ω
17                 β ← Σ_i ( x̄[i] · M_coeff(c̄[i])) + M_coeff(k);
18                 π ← φ[β/f(τ̄)] ∧ mbp
19                 preconds ← preconds ∪ {π};
20                 branches ← branches[π ↦ β];
21 пусть ξ новая переменная;
22 для всех π ∈ preconds
23     ξ ← ite(π, branches(π), ξ)
24 вернуть ⟨реализуема, ξ⟩;
```

ющие все значения входных параметров функции на строке 6, или в случае нереализуемости спецификации на строке 16.

В начале алгоритма множество предусловий и множество термов веток инициализируются пустыми множествами. На строке 5 все вхождения функции f вместе с её аргументами заменяются на новую перемен-

ную y для того, чтобы по полученной формуле построить модельную проекцию на строке 8 с помощью процедуры GetMBP.

Внутри основного цикла 6 на строке 7 получаем модель для модельной проекции, которая ещё не покрыта уже найденными предусловиям. На первой итерации цикла выбирается произвольная модель, поскольку множество предусловий пусто. Этот цикл перебирает модельные проекции, число которых по определению конечно. Внутренний цикл на строке 9 строит непосредственно ветки для спецификации, усиленной модельной проекцией, как это описано ранее. Цикл продолжается, пока найденные предусловия не покроют полученную модельную проекцию mbp.

После получения модели для входных параметров синтезируемой функции, все универсально квантифицированные переменные $\bar{x}$ заменяются на конкретные значения на строке 5. После чего подставляется шаблон терма ветки (9) вместо вызова функции f в исходной спецификации на строке 15, тем самым получив формулу со свободными переменными, которые представляют собой неизвестные коэффициенты. Для нахождения конкретных значений на строке 16 ищется модель для коэффициентов M_{coeff}.

На строках 25 и 18 вычисляются терм и предусловие новой ветки.

После нахождения всех веток строится терм для реализации функции $f(\bar{x})$, начиная со строки 20.

3.4. Пример

В качестве примера рассматривается спецификация для функции максимума из двух чисел (2). Тогда после замены вызова функции $max2$ справедлива формула (11). Для рассмотренной ранее модели первая модельная проекция будет выглядеть так: $x < y$.

Так как множество предусловий пусто, то модель для входных параметров функции будет $M_x = \{x \mapsto 1; y \mapsto 2\}$. Подставив их, получим

следующее:

$$max2(1,2) \geq 1 \wedge max2(1,2) \geq 2 \wedge (max2(1,2) = 1 \vee max2(1,2) = 2).$$

Затем подставим шаблон (9) вместо вызова $max2$ в исходную спецификацию:

$$\bar{c}[1] * 2 + \bar{c}[2] * 2 + k \geq 1 \wedge \bar{c}[1] * 1 + \bar{c}[2] * 2 + k \geq 2$$
$$\wedge (\bar{c}[1] * 1 + \bar{c}[2] * 2 + k = 1 \vee \bar{c}[1] * 1 + \bar{c}[2] * 2 + k = 2).$$

Значения неизвестных коэффициентов можно найти с помощью модели полученной формулы: $M_{coeff} = \{\bar{c}[0] \mapsto 0; \bar{c}[1] \mapsto 1; k \mapsto 0\}$. Тогда новая ветка будет определяться термом $\beta_1 \leftarrow y$ и соответствующим предусловием $\pi_1 \leftarrow x < y$.

Аналогично на новой итерации цикла 6, взяв модель $M = \{x \mapsto 1; y \mapsto 0\}$ для модельной проекции, можно получить вторую ветку $\langle x \geq y; x \rangle$.

4. Синтез по спецификациям с множественными вызовами

Спецификации с множественным вызовами выразительнее спецификаций с одиночным вызовом.

При этом не для всех спецификаций терм синтезируемый функции можно выразить через сигнатуру той же теории, в которой определена спецификация. Например, для спецификации в линейной арифметики $\exists f.\forall x.f(x+10) = f(x) \wedge (x > 0 \wedge x <= 10 \rightarrow f(x) = x)$ реализации функции f нельзя выразить в сигнатуре этой же теории.

В этой главе предполагается расширить ранее рассмотренную процедуру из главы 3 для синтеза по спецификациям с множественными вызовами. Предложенная процедура так же применима и для спецификаций с одиночным вызовом, так как такие спецификации является частным случаем спецификаций с множественными вызовами. Синтез по таким спецификациям можно условно разбить на две части: доказательство нереализуемости спецификации и поиск ветвей.

4.1. Доказательство нереализуемости спецификации

В отличие от спецификаций с одиночным вызовом, где доказательство нереализуемости функции можно свести к формуле первого порядка путем замены вызова на новую переменную, нереализуемость спецификаций с множественными вызовами задача нетривиальная.

Для того, чтобы доказать нереализуемость синтезируемой функции f, достаточно найти значения универсально квантивицированных переменных f в спецификации (1), при которых формула φ невыполнима независимо от реализации f. Вызов $f(\bar{x})$ после подстановки вместо $\bar{x}$ некоторых конкретных значений $\bar{v}$ называется конкретным вызовом.

Рассмотрим следующую спецификацию:

$$\exists f\,.\,\forall x, y\,.\,f(x) > f(x-1) \wedge f(y) > f(y+1).$$

Здесь переменные $\bar{x} = \langle x, y\rangle$. После замены $\bar{x}$ на конкретные значения

$\bar{v} = \langle 1, 0 \rangle$, бескванторная часть спецификации $f(1) > f(0) \wedge f(0) > f(1)$, становится невыполнимой независимо от интерпретации f.

Ключевая эвристика для ускорения поиска таких значений — это сокращение числа конкретных вызовов. Таким образом, мы приравниваем пары различных операндов f и извлекаем модели из сгенерированных равенств. Например, в вышеприведенной спецификации равенство $x = y + 1$ создается путем приравнивания операндов $f(x)$ и $f(y+1)$, которые появляются в спецификации. Отображение $M_1 = \{x \mapsto 1, \mapsto 0\}$ является моделью этого равенства. Точно так же дополнительно к приведенной спецификации можно рассматривать ограничение в виде одного из равенств $x = y$, $x - 1 = y + 1$ или $x - 1 = y$, что приводит к трем независимым проверкам выполнимости (соответственно: выполнимо, выполнимо и невыполнимо), и наличие хотя бы одной невыполнимой формулы гарантирует нереализуемость функции.

Если ни одно из рассмотренных равенств (или выполнимых конъюнкций равенств) не помогло, то можно попытаться перечислить различные возможные значения $\bar{x}$, подставить их в φ и проверить на невыполнимость.

Теорема 4.1. *Для спецификации $\exists f. \forall \bar{x}. \varphi(f, \bar{x})$ и выполнимой формулы ψ, если $\varphi \wedge \psi$ невыполнима, то тогда спецификации нереализуема.*

Для синтеза по спецификациям с множественными вызовами эффективный способ получить ψ и попытаться использовать его далее в теореме 4.1 — это собрать операнды каждого вызова и приравнять их. Таким образом, по сравнению с синтезом по спецификациям с одиночным вызовом (который обычно считается более простой задачей), можно использовать отличительное свойство, заключающееся в том, что функция f вызывается несколько раз с различными аргументами.

Далее множество аргументов (операндов) f в каждом вхождении f в φ будет обозначаться как $ops(f, \varphi)$ (или для краткости $ops(f)$, когда это будет ясно из контекста). Основная задача для доказательства нереализуемости — это создание всевозможных попарных равенств для элементов $ops(f)$. Например, предположим, что функция f ар-

ности один и множество $eqs(ops(f))$ является $\{a = b \mid a, b \in ops(f)$ и $a \neq b\}$. Таким же образом в общем случае для f с арностью n, $eqs(ops(f)) \stackrel{\text{def}}{=} \{a_1 = b_1 \wedge \ldots \wedge a_n = b_n \mid (a_1, \ldots, a_n), (b_1, \ldots, b_n) \in ops(f)$ and $a \neq b\}$.

Процедура доказательства нереализуемости спецификации поддерживает множество формул (ограничения реализуемости) $eqs(ops(f))$ и итеративно рассматривает все подмножества $U \in 2^{eqs(ops(f))}$ (включая пустое множество), такие что конъюнкция $\bigwedge\limits_{\alpha \in U} \alpha$ выполнима. Тогда по теореме 4.1, если формула $\varphi \wedge \bigwedge\limits_{\alpha \in U}$ невыполнима, то тогда спецификация нереализуема.

Автоматическая процедура определения нереализуемости встроена в основную процедуру синтеза программ по спецификациям с множественными вызовами.

4.2. Поиск ветвей синтезируемой функции с множественными вызовами

Поиск терма некоторой ветви осуществляется тем же образом, как и в случае с одиночным вызовом, с помощью шаблона (9). Так как функции f имеет несколько различных вызовов, то для поиска такой ветви можно рассмотреть две разных стратегии.

Первая стратегия предполагает, что все вызовы в спецификации следуют внутри одной и той же ветви, что позволяет использовать один и тот же шаблон (9) для каждого вызова. Таким образом, такая стратегия вводит меньше коэффициентов, тем самым уменьшая размер создаваемой спецификации, а также пространство поиска для коэффициентов. Несмотря на недостаток полноты, эта стратегия эффективна на практике. Алгоритм гарантирует корректность, явно проверяя, есть ли необходимость создания для какого-либо конкретного входа новой ветви, и выполняет (следующую) итерацию.

Вторая стратегия предусматривает более широкую область поиска и генерирует несколько веток одновременно. Основные идеи в обеих стратегиях схожи, однако для некоторых спецификаций только конкретная

стратегия может обнаружить реализацию.

Задачу нахождения наиболее общих предусловий при разных вызовах можно рассматривать как проблему мультиабдукции. В случае первой стратегии, когда рассматривается только одна ветка $\langle \pi_i, \beta_i \rangle$, решается проблема нелинейной мультиабдукции, которую можно формализовать так:

$$\bigwedge_{\bar{\tau} \in ops(f,\varphi)} \pi_i(\bar{\tau}) \implies \varphi[\beta_i(\bar{\tau})/f(\bar{\tau})...]_{\bar{\tau} \in ops(f,\varphi)}. \tag{12}$$

При такой постановке проблемы π_i рассматривается как неизвестный предикат. Он должен быть нетривиальным, другими словами, отличным от $false$. При этом π_i должен быть по возможности наиболее общим, т.е. для любого другого возможного предиката $\hat{\pi}_i$, для которого справедливо выражение (12), должно выполняться $\hat{\pi}_i \implies \pi_i$.

4.3. Абдукция

Для поиска предусловий предполагается использовать нелинейную мультиабдукцию. Мультиабдукция является обобщение простой (стандартной) абдукции [12]. Пусть дана пара формул (χ, C) и неизвестный предикат $R(x)$, тогда стандартная абдукция находит такую интерпретацию ϕ для R, что $\phi \wedge \chi \not\models \bot$ и $\phi \wedge \chi \models C$. Иначе говоря, стандартная абдукция ищет некоторую объясняющую гипотезу ϕ, которая вместе с известными фактами χ влечет заключение C, т.е. $R(x) \wedge \chi \implies C$.

Например, нам известно, что $x \geq -2$, и мы хотим доказать $x+y > 10$. Тогда искомой гипотезой ϕ будет $y > 12$.

Простой процедурой поиска объясняющей гипотезы ϕ для теорий первого порядка, допускающих устранение квантора, будет процедура устранения квантора из формулы $\forall \bar{y} \chi \implies C$, где $\bar{y}$ – все свободные переменные в $\chi \implies C$, кроме x.

Задача вывода неизвестных предикатов для более общего случая, когда неизвестных предикатов больше одного или вхождений одного и того же предиката больше одного, является задачей мультиабдук-

ции [1]. В общем виде такую задачу можно представить следующим образом:

$$(\bigwedge_{i=1}\bigwedge_{j=1} R_i(x_{ij}) \wedge \chi) \to C. \tag{13}$$

При этом требуется отыскать интерпретации для неизвестных предикатов R_i.

Случай, когда существует несколько вхождений одного и того же предиката с различными аргументами $x_1, ..x_n$, называется нелинейной мультиабдукцией, в противном случае — линейной. Для решений подобных задач в статье [1] предложена эвристическая процедура. В частности, для решения задач линейной мультиабдукции используется декартовая декомпозиция. Для нелинейной мультиабдукции — изоморфная декомпозиция.

4.4. Алгоритм синтеза, использующий одну ветку

Псевдокод процедуры для синтеза программ по спецификациям с одиночным вызовом представлен в листинге 3. Представленная процедура является расширением ранее рассмотренного алгоритма синтеза, псевдокод которого приведен в листинге 2, на случай спецификаций с множественными вызовами.

Входные и выходные параметры процедуры совпадают с процедурой для синтеза по спецификациям с одиночным вызовом.

Одно из незначительных отличий заключается в построении модельных проекций для ограничения спецификации. На строке 3 инициализируется множество всех найденных проекций. Для построения проекций в исходной спецификации все различные вызовы функции заменяются на соответствующие переменные $\bar{y}$ в цикле 6. Соответственно, модельная проекция строится по модели, полученной из отрицания уже найденных проекций, и переменным $\bar{y}$.

Другое отличие заключается в нахождении модели M_x для квантифицированных переменных x на строке 13. Дизъюнкция в условии цикла для нахождения модели M_x отличается от условия завершения

Листинг 3: Алгоритм синтеза по спецификациям с множественными вызовами, использующий одну ветку OneBranchMISynt($\exists f.\forall \bar{x}.\varphi(f,\bar{x})$)

Вход: $\exists f.\forall \bar{x}.\varphi(f,\bar{x})$

Выход: $res \in \{$(не)реализуема$\}$, терм реализации для $f(\bar{x})$

$preconds \leftarrow \varnothing$;
$branches \leftarrow \varnothing$;
$mbps \leftarrow \varnothing$;
$i \leftarrow 0$;
пусть $\bar{y}$ новые переменные;
для всех $\bar{\tau} \in ops(f,\omega)$
  $\varphi_y \leftarrow \varphi[\bar{y}[i]/f(\bar{\tau})]$;
  $i \leftarrow i+1$;
пока $\models \bigwedge_{\pi \in preconds} \neg\pi(\bar{x})$
  $M_{mbp} \models \varphi_y(y,\bar{x}) \wedge \bigwedge_{\mu \in mbps} \neg\mu(\bar{x})$;
  $mbp \leftarrow GetMBP(\bar{y}; M_{mbp}; \exists \bar{y}.\varphi_y)$;
  $mbps \leftarrow mbps \cup \{mbp\}$;
  пока $\exists M_x.M_x \models \bigvee_{\bar{\tau} \in ops(f,\varphi)} \Big(\bigwedge_{\pi \in preconds} \neg\pi(\bar{\tau}) \Big) \wedge mbp$
    $\omega \leftarrow \varphi[M_x(\bar{x})/\bar{x}]$;
    если $\not\models \omega$
      вернуть $\langle$нереализуема$, \varnothing\rangle$;
    иначе
      пусть $\bar{c}$ и k новые переменные;
      для всех $\bar{\tau} \in ops(f,\omega)$
        если $\exists \pi \in preconds\,.\,\pi(\bar{\tau}) = \top$
          $\omega \leftarrow \omega[branches(\pi)(\bar{\tau})/f(\bar{\tau})]$;
        иначе
          $\omega \leftarrow \omega\big[\sum_i (\bar{\tau}[i] \cdot \bar{c}[i]) + k/f(\bar{\tau})\big]$;
      если $\exists M_{coeff}\,.\,M_{coeff} \models \omega$
        $\beta \leftarrow \sum_i \big(\bar{x}[i] \cdot M_{coeff}(\bar{c}[i])\big) + M_{coeff}(k)$;
        $\pi \leftarrow Abduce(\bigwedge_{\bar{\tau} \in ops(f,\varphi)} \pi(\bar{\tau}) \rightarrow \varphi[\beta(\bar{\tau})/f(\bar{\tau})...]_{\bar{\tau} \in ops(f,\varphi)} \wedge mbp)$
        $preconds \leftarrow preconds \cup \{\pi\}$;
        $branches \leftarrow branches[\pi \mapsto \beta]$;
      иначе
        вернуть SimulBranchMISynt$(f, \varphi, \bar{x}, M_x)$;
пусть ξ новая переменная;
для всех $\pi \in preconds$ $\xi \leftarrow ite(\pi, branches(\pi), \xi)$;
вернуть $\langle$реализуема$, \xi\rangle$;

процедуры на строке 9 и является более слабой. Внутри цикла проверяется реализуемость для конкретного вызова, полученного по модели M_x.

В случае реализуемости все полученные конкретные вызовы принадлежат двум группам: 1) вызовам, которые удовлетворяют некоторым уже вычисленным предусловиям, и 2) вызовам, которые не удовлетворяют никаким ранее полученным предусловиям. Для первой группы алгоритм использует соответствующую ветвь (строка 21), в противном случае переходит к новому шаблону ветки (строка 23). Таким образом, для первой группы переиспользуются уже найденные до этого ветки.

После построения терма ветки на 25 строке ищется предусловие. Существенным отличием от алгоритма, использующего одну ветку, является то, что предусловие ищется с помощью процедуры ABDUCE на 28 строке, которая решает проблему абдукции и возвращает реализацию для неизвестного предиката π. Возможный псевдокод такой процедуры представлен в статье [1]. Абдукция применяется к исходной спецификации, усиленной модельной проекцией, с заменой вызовов функции на терм синтезируемой ветки.

Если формула является невыполнимой в строке 24, то на этой итерации должно быть определено более одной ветви одновременно. Соответственно, в таком случае требуется рассмотреть более общую стратегию синтеза, использующую не одну ветку, а несколько на одной итерации.

В качестве примера рассмотрим следующую задачу синтеза:

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

После замены вызовов на новые соответствующие переменные $\hat{y}[0]$ и $\hat{y}[1]$ получается формула φ_y: $\exists \bar{\hat{y}} \,.\, \forall x, y \,.\, \hat{y}[0] + \hat{y}[1] \geq x - y$, чья модельная проекция дает $true$ для любой модели φ_y.

На первой итерации внутреннего цикла предложенной процедуры модель $M_x = \{x \mapsto 1, y \mapsto 0\}$, далее после замены переменных $\bar{x}$ на значения из модели: $\varphi[M_x(\bar{x})/\bar{x}] = f(1) + f(0) \geq 1$, $\omega = \bar{c}[0] \cdot 1 + \bar{c}[1] \cdot 1 + 2 \cdot k \geq 1$, тогда модель для получения коэффициентов $M_{coeff} = \{\bar{c}[0] \mapsto 1, \bar{c}[1] \mapsto 0, k \mapsto 0\}$ позволяет найти терм ветки $\beta_1(x) = x$.

Получив терм ветки, предусловие для ветки $\beta_1(x)$ вычисляется как предикат $\pi_1(x)$, для которого справедливо следующее:

$$\pi_1(x) \wedge \pi_1(y) \implies x + y \geq x - y.$$

Возможное решение проблемы мультиабдукции выглядит так: $\pi_1(x) = x \geq 0$.

На второй итераций алгоритм находит модель $M'_x = \{x \mapsto 1, y \mapsto -1\}$. Однако, так как предусловие $\pi_1(x)$ уже найдено, можно использовать терм ветки β_1 вместо вызова $f(1)$ и ввести шаблон ветки только для конкретного вызова $f(-1)$, который специфицирован ниже.

$$\omega = f(1) + f(-1) \geq 2$$

$$1 + \bar{c}[0] \cdot (-1) + k \geq 2$$

Тогда для модели коэффициентов $M'_{coeff} = \{\bar{c}[0] \mapsto -1, k \mapsto 0\}$ существует терм $\beta_2(x) = -x$. Соответствующее предусловие $\pi_2(x) = x \leq 0$ находится по мультиабдукции:

$$\pi_2(x) \wedge \pi_2(y) \implies -x - y \geq x - y.$$

Таким образом, возможное решение рассмотренной проблемы синтеза выглядит так:

$$f(x) = ite(x \geq 0, x, -x).$$

Реализация имеет две ветки $\beta_1(x) = x$ and $\beta_2(x) = -x$ с двумя предусловиями $\pi_1(x) = x \geq 0$ и $\pi_2(x) = \neg(x \geq 0)$.

4.5. Алгоритм синтеза, использующий несколько веток одновременно

Листинг 4 содержит псевдокод алгоритма, позволяющего синтезировать несколько ветвей одновременно. Он следует принципу поиска веток на основе модели, аналогичному идеи в алгоритме, использующем одну ветку, и выполняет итерации до тех пор, пока не будут найдены

Листинг 4: Алгоритм синтеза по спецификациям с множественными вызовами, использующий несколько веток одновременно SimulBranchMISynt($f, \varphi, \bar{x}, M_x, preconds, branches$)

Вход: $\exists f \,.\, \forall \bar{x} \,.\, \varphi(f, \bar{x})$, модель M_x такая, что $M_x \models \varphi$, предикаты $preconds$, отображение $branches$ предикатов в термы

Выход: $res \in \{$(не)реализуема$\}$, терм реализации для $f(\bar{x})$

$\omega \leftarrow \varphi[M_x(\bar{x})/\bar{x}]$;
если $\omega \implies \bot$ **вернуть** $\langle$нереализуема, $\varnothing\rangle$;
$used_\pi \leftarrow \varnothing$;
$coeffs \leftarrow \varnothing$;
for $\bar{\tau} \in ops(f, \omega)$ **do**
    если $\exists \pi \in preconds \,.\, \pi(\bar{\tau}) = \top$
        $used_\pi \leftarrow used_\pi \cup \{\pi\}$;
        $\omega \leftarrow \omega[branches(\pi)(\bar{\tau})/f(\bar{\tau})]$;
    иначе
        пусть $\bar{c}$ и k новые переменные;
        $coeffs \leftarrow coeffs \cup \{\langle \bar{c}, k\rangle\}$;
        $\omega \leftarrow \omega\big[\sum\limits_i (\bar{\tau}[i] \cdot \bar{c}[i]) + k / f(\bar{\tau})\big]$;
end
если $\exists M_{coeff} \,.\, M_{coeff} \models \omega$
    $new_\beta \leftarrow \Big\{ \sum\limits_i \big(\bar{x}[i] \cdot M_{coeff}(\bar{c}[i])\big) + M_{coeff}(k) \;\Big|\; \langle \bar{c}, k\rangle \in coeffs \Big\}$;
    если $\exists new_\pi \,.\, img(new_\pi) = new_\beta \wedge dom(new_\pi) \cap preconds = \varnothing$ **и** (15) выполняется для $preconds \cup dom(new_\pi)$ and $branches \cup new_\pi$
        $preconds \leftarrow preconds \cup dom(new_\pi)$;
        $branches \leftarrow branches \cup new_\pi$;
иначе
    вернуть SimulBranchMISynt($f, \varphi, \bar{x}, M_x, preconds \setminus used_\pi, branches$);
пусть ξ новая переменная;
для всех $\pi \in preconds$
    $\xi \leftarrow ite(\pi, branches(\pi), \xi)$;
если $\exists M_3 \,.\, M_3 \models \neg\varphi[\xi/f]$
    вернуть SimulBranchMI($f, \varphi, \bar{x}, M_3, preconds, branches$);
иначе
    вернуть $\langle$реализуема, $\xi\rangle$;

все ветви. Алгоритм также способен доказать нереализуемость.

Алгоритм является рекурсивным, и он принимает модель M_x в качестве входных данных. Первоначально M_x может быть сгенерирован алгоритмом, использующем одну ветку, недетерминированным или даже предложенным пользователем способом. После генерации каждого множества ветвей и предусловий алгоритм ищет новую модель, которая еще не покрывает всевозможных входных значений функции, и использует ее для рекурсивного вызова (строка 25). Ветви и предусловия распределяются между рекурсивными вызовами, что позволяет лениво создавать их и повторно использовать при необходимости.

Основное отличие от алгоритма, использующем одну ветку, заключается в построении конкретизированной спецификации ω: для каждого различного конкретного вызова, который не имеет своей собственной ветви, алгоритм вводит отдельный шаблон (строка 12) и далее создает ветку (строка 15).

Может случиться так, что некоторые из конкретных вызовов охватываются предусловиями, приведенными в качестве входных данных. Как и в алгоритме, использующем одну ветку, предусловия повторно используются в алгоритме, использующем несколько веток (строка 8). Однако это может привести к невыполнимым формулам. В таком случае алгоритм отбросит предусловие и рекурсивно (строка 19) вызовет себя.

Формула ω может быть невыполнимой (строка 19), только если множество $used_\pi$ непусто (поскольку в каждой выполнимой формуле $\varphi[M_x(\bar{x})/\ vx]$ вызовы f можно заменить новыми целочисленными переменными, значения которых определяют константы c в шаблонах термов ветвей).

Предусловия создаются аналогично тому, как они создаются в алгоритме, использующем одну ветку. Единственное отличие состоит в том, что мы проверяем сразу несколько различных предусловий (строка 16) одновременно. Для этого вводится $Comb(S, D)$ — множество всех возможных отображений из конечного множества S в конечное множество

D (размещения). Например,

$$Comb(\{1,2\},\{1,2,3\}) \stackrel{\text{def}}{=} \Big\{\{1\mapsto 1, 2\mapsto 1\}, \{1\mapsto 1, 2\mapsto 2\}, \{1\mapsto 1, 2\mapsto 3\},$$
$$\{1\mapsto 2, 2\mapsto 1\}, \{1\mapsto 2, 2\mapsto 2\}, \{1\mapsto 2, 2\mapsto 3\},$$
$$\{1\mapsto 3, 2\mapsto 1\}, \{1\mapsto 3, 2\mapsto 2\}, \{1\mapsto 3, 2\mapsto 3\}\Big\}$$

В случае с K различными наборами аргументов, например, $ops(f,\varphi) = \{\bar{\tau}_1,\ldots,\bar{\tau}_K\}$ и N предусловиями/ветками, например, для $preconds = \{\pi_1,\ldots,\pi_N\}$ и $\beta_i = branches(\pi_i)$ должны выполняться условия:

$$\forall \pi_i \,.\, \pi_i(\bar{x}) \;\not\Longrightarrow\; \bot \tag{14}$$

$$\forall m \in Comb(ops(f,\varphi),\{1,\ldots N\})\,. \tag{15}$$
$$\bigwedge_{1\le i\le K} \pi_{m(\bar{\tau}_i)}(\bar{\tau}_i) \;\Longrightarrow\; \varphi[\beta_{m(\bar{\tau}_1)}(\bar{\tau}_1)/f(\bar{\tau}_1)]\ldots[\beta_{m(\bar{\tau}_K)}(\bar{\tau}_K)/f(\bar{\tau}_K)]$$

Условие (14) является ограничением нетривиальности предикатов. Система ограничений (15) состоит в общем случае из N^K задач мультиабдукций. Решение всех задач может быть слишком затратно для случаев многих вызовов и многих ветвей. Однако на практике часто бывает достаточно решить одну более сильную проблему и проверить результат по другим более слабым (простым) проблемам.

В псевдокоде через dom и img обозначаются прообраз и образ отображения (например, new_β, который связывает введенные новые предусловия и ветви).

В качестве примера рассмотрим задачу синтеза реализации для f, которая удовлетворяет следующей спецификации:

$$\exists f \,.\, \forall \bar{x} \,.\, f(x) = f(-x) \wedge f(x) \geq x.$$

Для синтеза вызывается процедура $OneBranchMISynt$, псевдокод которой приведен в листинге 3. Очевидно, что после замены всех вызовов

на новые переменные любая модельная проекция полученной формулы будет $true$. На первой итерации алгоритм найдет модель $\{x \mapsto 0\}$, по которой можно получить ветку с предусловием $\pi_1(x) = x \leq 0$ и термом $\beta_1(x) = 0$. Поскольку справедливо следующее выражение:

$$x \leq 0 \wedge -x \leq 0 \implies 0 = 0 \wedge 0 \geq x$$

на следующей итерации алгоритм найдет такую модель для входных параметров функции $\{x \mapsto 1\}$, при которой конкретизированная спецификация $\omega = f(1) = f(-1) \wedge f(1) \geq 1$. Поскольку выходное значение функции -1, ветка уже найдена, вводятся новые коэффициенты только для вызова $f(1)$:

$$\bar{c}[0] \cdot 1 + k = 0 \wedge \bar{c}[0] \cdot 1 + k \geq 1.$$

Эта формула невыполнима и тогда вызывается процедура для синтеза, использующая несколько веток одновременно.

Для двух конкретных вызовов $f(1)$ и $f(-1)$ и пустого множества предусловий алгоритм вводит два новых набора переменных для коэффициентов (c_1, k_1 и c_2, k_2), и тогда спецификация ω определяется как:

$$c_1 \cdot 1 + k_1 = c_2 \cdot (-1) + k_2 \wedge c_1 \cdot 1 + k_1 \geq 1.$$

Эта формула выполнима с такой моделью $M_{coeff} = \{c_1 \mapsto 1, c_2 \mapsto -1, k_1 \mapsto 0, k_2 \mapsto 0\}$. Данная формула позволяет найти следующие ветки: $\beta_1(x) = x$, и $\beta_2(x) = -x$. Для нахождения предусловий решается система задач мультиабдукции:

$$\pi_1(x) \wedge \pi_2(-x) \implies x = -x \wedge x \geq x$$

$$\pi_2(x) \wedge \pi_1(-x) \implies -x = x \wedge -x \geq x$$

В конечном итоге получается следующее решение:

$$f(x) = ite(x \geq 0, x, -x).$$

5. Инструмент

5.1. Архитектура

Алгоритм для синтеза по спецификациям с множественными вызовами, который обобщает алгоритм для синтеза по спецификациям с одиночным вызовом, был реализован в виде инструмента для синтеза (другими словами, синтезатора). Реализованный программный инструмент позволяет по спецификации, которая представляет собой логическую формулу теории линейной целочисленной арифметики, в формате Synth-Lib как с множественными, так и одиночными вызовами генерировать реализацию функции в виде терма с если-тогда-иначе оператором. В случае нереализуемости спецификации инструмент сообщает об этом.

Инструмент был реализован в виде расширяющего модуля Z3-решателя на языке C++. Исходный код всего модуля занимает в общей сложности 4175 строк.

На вход синтезатору подается файл формата .sl, содержащий задачу синтеза в формате Synth-Lib. На выходе определение функции в соответствующем формате. Пример вывода для функции максимума из двух чисел:

```
(define-fun max2 ((x Int) (y Int))  Int (ite (>= x y) x y)))
```

Поддержка других команд Synth-Lib языка была реализована в Z3 на основе формата SMT-Lib 2 [5]. Большинство остальных команд Synth-Lib (например, define-fun, let и т.д.) поддерживается самим языком SMT-Lib 2, который в Z3 реализован изначально. Таким образом, с помощью решателя Z3 осуществляется разбор входного файла в набор ограничений, требующихся для синтеза.

Синтезатор реализует процедуру, описанную в листинге 3, что позволяет синтезировать программы по спецификациям как с одним вызовом, так и с множественными вызовами. Синтезатор использует возможности Z3 для решения SMT-задач, в частности, получения моделей, модельных проекций (процедура GetMBP), а также для устранения

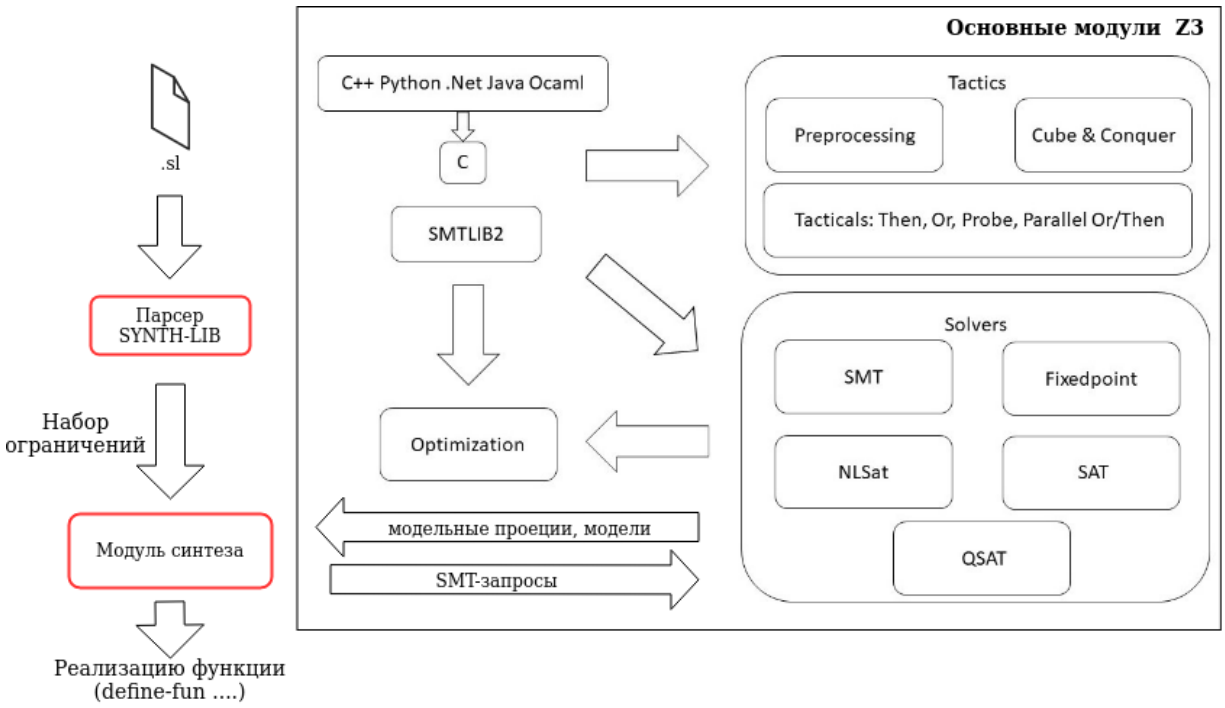

Рис. 2: Реализованный инструмент в инфраструктуре Z3

квантора, обращаясь напрямую к соответствующим модулям. На рисунке 2 представлена общая архитектура Z3 вместе с реализованным инструментом.

5.2. Реализацию расширяющего модуля для Z3-решателя

На рисунке 3 представлена диаграмма компонентов модуля синтеза для Z3-решателя. Компоненты OneBranchMISynt и SimulBranchMISynt реализуют псевдокод листинга 3 и псевдокод листинга 4. Оба компонента зависят от процедуры ABDUCE, которая реализует мультиабдукцию, описание которой представлено в разделе 4.3.

5.3. Реализация эвристик

Успешность синтеза во многом зависит от найденных коэффициентов для терма ветки в шаблоне (9). Для эффективного поиска коэффициентов реализован ряд эвристик:

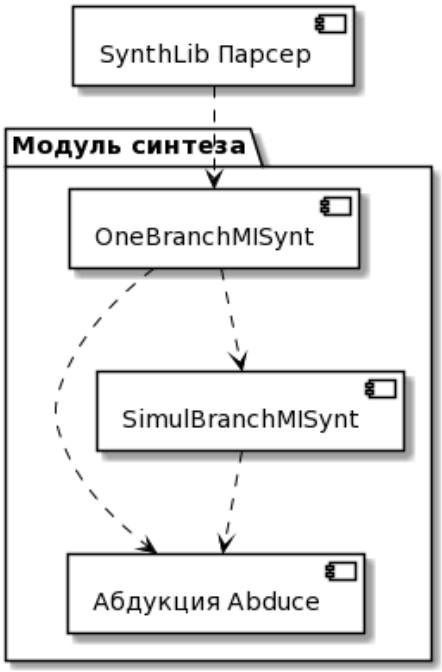

Рис. 3: Диаграмма компонентов

Чёрный список моделей. Один и тот же терм ветви может быть синтезирован много раз по алгоритму, но если он не дает достаточно общего предусловия, то не имеет смысла рассматривать его снова. Таким образом, уже рассмотренные модели для коэффициентов M_{coeff} добавляются в черный список и больше не генерируются.

Честность перебора коэффициентов. Данная эвристика позволяет не зацикливаться на переборе одного и того же коэффициента, поскольку SMT-решатель, возвращающий модель коэффициентов, не дает никаких гарантий на перебор значений. Это реализовано с помощью набора экземпляров (в количестве коэффициентов) решателей, ответственного за перебор значений своего коэффициента. При этом на каждой итерации запрос осуществляется к одному из решателей по очереди, т.е. на следующей итерации запрос будет к следующему в наборе решателю. Это позволяет равномерно перебирать значения коэффициентов.

Предпочтение простых моделей. Из-за недетерминированности в SMT решателях значения коэффициентов и констант термов ветки

могут быть большими и нелогичными. Иногда фактические значения, необходимые в реализации, могут быть получены из спецификации. Кроме того, сначала стоит рассмотреть коэффициенты и константы из тривиального набора значений: $\{-1, 0, 1\}$.

Однако если оказывается, что эти константы бесполезны, алгоритм ослабляет это ограничение и переходит к генерации произвольных моделей.

Коэффициенты для ветки из литералов. На практике часто найти удачные ветки можно с помощью коэффициентов, полученных из литералов. Например, для литерала с равенством $max2(x, y) = x$ можно получить следующее ограничение на коэффициенты: $C_0 = 0 \wedge C_1 = 1 \wedge C_2 = 0$.

Для решения системы задач мультиабдукции (15) предполагается решить сначала одну из простых задач (с меньшим количеством вхождений неизвестных предикатов), а затем полученное решение, т.е. найденные предикаты, проверить на остальных задачах. Проверку можно осуществить, подставив найденные предикаты в импликацию и проверив на истинность. В случае неудачи попытаться решить следующую задачу абдукции.

Кроме представленных эвристик для поиска коэффициентов, применяется эвристика для уменьшение веток синтезируемой программы:

Поглощение веток. Так как техника генерации ветвей ленива и основана на моделях, трудно оценить качество получаемых формул. Возможно, в следующих итерациях алгоритм генерирует более слабое предусловие, предлагая альтернативную ветвь некоторой существующей. Для этого выполняется проверка подстановки, чтобы определить такие случаи и, следовательно, сделать финальные решения более компактными. Проверка заключается в запросе SMT-решателю: для любых двух веток i и j, если истинно $\pi_i(\bar{x}) \implies \pi_j(\bar{x})$, то ветка j удаляется.

6. Экспериментальное исследование

Оценка полученной реализации была проведена по спецификациям с множественными вызовами, взятых из соревнования синтезаторов SyGuS-Comp[3], сложным спецификациям, созданным вручную, а также по спецификациям с одиночным вызовом из SyGuS-Comp.

В качестве конкурирующего инструмента рассматривался CVC4, поскольку он является алгоритмически самым близким и самым недавно разработанным инструментом. CVC4 реализует комбинацию синтеза, основанного на опровержении (раздел 2.7.1), и перечислительного синтеза (раздел 2.5) [13].

Каждое решение инструмента MI-SYNT было проверено с помощью подстановки реализации синтезируемой функции в спецификацию, реализовано это путем дополнительного SMT-запроса решателю с ограничением $\forall \bar{x}.f(\bar{x}) = \xi$, где ξ — реализации функции

6.1. Спецификации с одиночными вызовами

Инструмент MI-SYNT был запущен на 50 спецификациях с одиночным вызовом из соревнования SyGuS-Comp.

- 19 спецификаций для функции максимума max с количеством аргументов от 2 до 20.
- 17 задач синтеза для функции суммы $findSum$ — арности от 2 до 10.
- 14 задач синтеза для функции поиска индекса ближайшего элемента $findIdx$ — арности от 2 до 15.

CVC4 и MI-SYNT оба решили все 50 спецификаций за сравнительно одинаковое время (отличие в несколько долей секунд).

[3]https://sygus.org/

6.2. Спецификации с множественными вызовами

Было рассмотрено 87 спецификаций с множественными вызовами: 41 нереализуемых и 46 реализуемых.

Общая сравнительная характеристика инструментов CVC4 и MI-SYNT на спецификациях с множественными вызовами представлена в таблице 1.

Синтезатор	Нереализуемые спецификации (41)		Реализуемые спецификации (46)	
	Решил	Быстрее	Решил	Быстрее
MI-SYNT	34	8	35	26
CVC4	33	5	24	7

Таблица 1: Результаты сравнения CVC4 и MI-SYNT

6.2.1. Нереализуемые спецификации

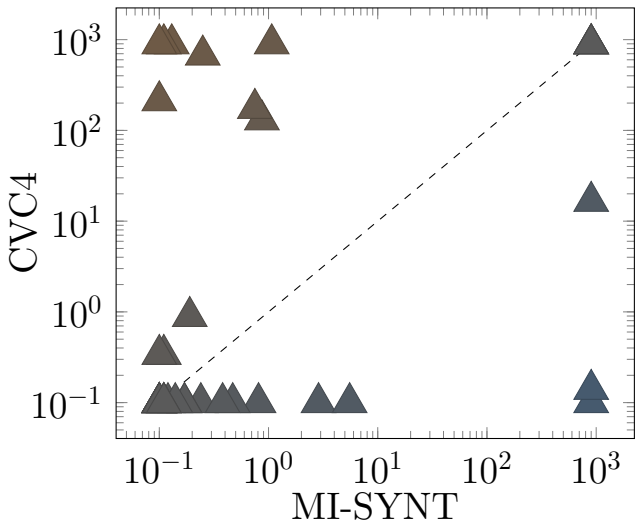

Рис. 4: Сравнение MI-SYNT и CVC4 на нереализуемых спецификациях. Каждый треугольник на графике представляет пару времен выполнения (сек × сек). Таймауты размещены на границах.

В течение 900 секунд MI-SYNT решил 34 из 41 нереализуемых спецификаций (в то время как CVC4 решил 33). Время выполнения MI-SYNT находится в диапазоне от долей секунды до 5,53 с (в то время

как для CVC4 до 676,49 с). Из 41 эталонного теста 17 заняло нетривиальное время (более 1 секунды) обоими синтезаторами: в 8 случаях MI-SYNT обошёл по времени исполнения CVC4 и в 5 случаях CVC4 не решил спецификации, на которых MI-SYNT завершился (остальные 4 не были решены ни одним из инструментов). Сравнительная статистика времени исполнения MI-SYNT и CVC4 представлена на рисунке 4.

6.2.2. Реализуемые спецификации

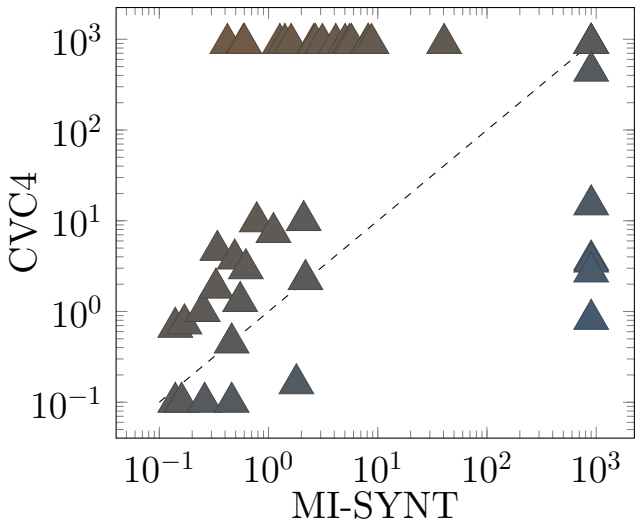

Рис. 5: Сравнение MI-SYNT и CVC4 на реализуемых спецификациях. Каждый треугольник на графике представляет пару времен выполнения (сек × сек). Таймауты размещены на границах.

Из 46 тестов MI-SYNT решил 35 (в то время как CVC4 решил 24). Время работы MI-SYNT варьировалось от доли секунды до 40,51 секунды (в то время как для CVC4: до 445,66 секунды). Более того, 38 тестов заняли нетривиальное время как по MI-SYNT, так и по CVC4: в 26 случаях MI-SYNT превзошел CVC4 по времени исполнения, и только в 7 случаях CVC4 превзошёл MI-SYNT. Сравнительная статистика времени исполнения MI-SYNT и CVC4 на реализуемых спецификациях показана на рисунке 5.

По не решенным инструментом CVC4 спецификациям можно сделать вывод, что CVC4 плохо работает на спецификациях, чьи реализации функций содержат большие значения коэффициентов или кон-

стант. Можно предположить, что основная причина этому — перечислительный метод, который не позволяет перебрать большие значения констант из-за плохой масштабируемости.

Заключение

В рамках работы были достигнуты следующие результаты.

- Описаны ключевые понятия функционального синтеза. Проанализированы существующие подходы к синтезу: функциональный синтез, синтаксически-управляемый синтез. Выявлены их недостатки и достоинства.

- Разработан алгоритм синтеза нерекурсивных функций по спецификациям теории линейной целочисленной арифметики с одиночным вызовом с использованием модельных проекций и подстановки вместо вызовов функции шаблонов (9) для термов.

- Разработан алгоритм синтеза нерекурсивных функций по спецификациям теории линейной целочисленной арифметики с множественными вызовами, использующий мультиабдукцию для поиска предусловий.

- Был реализован программный инструмент MI-SYNT как расширяющий модуль синтеза по спецификациям с с одиночными и множественными вызовами для решателя Z3 на языке программирования C++.

- Проведено экспериментальное исследование на 50 спецификациях с одиночным вызовом и 87 спецификациях с множественными вызовами, часть которых взята из SyGuS-Comp. MI-SYNT смог решить 69 спецификаций с множественными вызовами, в то время как известным синтезатор CVC4 смог решить только 55 задач, что доказывает конкурентоспособность реализованного инструмента.

Список литературы

[1] Albarghouthi Aws, Dillig Isil, Gurfinkel Arie. Maximal Specification Synthesis // Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — POPL '16. — New York, NY, USA : ACM, 2016. — P. 789–801. — Access mode: http://doi.acm.org/10.1145/2837614.2837628.

[2] Alur Rajeev, Cerny Pavol, Radhakrishna Arjun. Synthesis through Unification. — arXiv : cs.PL/http://arxiv.org/abs/1505.05868v1.

[3] Alur Rajeev, Radhakrishna Arjun, Udupa Abhishek. Scaling Enumerative Program Synthesis via Divide and Conquer // Tools and Algorithms for the Construction and Analysis of Systems. — 2017. — Jan. — Access mode: http://dx.doi.org/10.1007/978-3-662-54577-5_18.

[4] Android Testing via Synthetic Symbolic Execution / Xiang Gao, Shin Hwei Tan, Zhen Dong, Abhik Roychoudhury // Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. — ASE 2018. — New York, NY, USA : ACM, 2018. — P. 419–429. — Access mode: http://doi.acm.org/10.1145/3238147.3238225.

[5] The SMT-LIB Standard: Version 2.5 : Rep. / Department of Computer Science, The University of Iowa ; Executor: Clark Barrett, Pascal Fontaine, Cesare Tinelli : 2015. — Available at www.SMT-LIB.org.

[6] Bjørner Nikolaj, Janota Mikolás. Playing with Quantified Satisfaction. // LPAR (short papers). — 2015. — Vol. 35. — P. 15–27.

[7] Bodik Rastislav, Jobstmann Barbara. Algorithmic Program Synthesis: Introduction // Int. J. Softw. Tools Technol. Transf. — 2013. — Oct. — Vol. 15, no. 5–6. — P. 397–411. — Access mode: https://doi.org/10.1007/s10009-013-0287-9.

[8] Complete Functional Synthesis / Viktor Kuncak, Mikaël Mayer, Ruzica Piskac, Philippe Suter // SIGPLAN Not. — 2010. — Jun. — Vol. 45, no. 6. — P. 316–329. — Access mode: http://doi.acm.org/10.1145/1809028.1806632.

[9] Counterexample-Guided Quantifier Instantiation for Synthesis in SMT / Andrew Reynolds, Morgan Deters, Viktor Kuncak et al. // Computer Aided Verification / Ed. by Daniel Kroening, Corina S. Păsăreanu. — Cham : Springer International Publishing, 2015. — P. 198–216.

[10] David Cristina, Kroening Daniel. Program synthesis: challenges and opportunities // Philosophical transactions. Series A, Mathematical, physical, and engineering sciences. — 2017. — Oct. — Vol. 375, no. 2104. — P. 20150403. — 28871052[pmid]. Access mode: https://pubmed.ncbi.nlm.nih.gov/28871052.

[11] De Moura Leonardo, Bjørner Nikolaj. Satisfiability modulo Theories: Introduction and Applications // Commun. ACM. — 2011. — Sep. — Vol. 54, no. 9. — P. 69–77. — Access mode: https://doi.org/10.1145/1995376.1995394.

[12] Dillig Isil, Dillig Thomas. Explain: A Tool for Performing Abductive Inference // CAV. — Vol. 8044. — 2013. — P. 684–689.

[13] Extending enumerative function synthesis via SMT-driven classification / H. Barbosa, A. Reynolds, D. Larraz, C. Tinelli // 2019 Formal Methods in Computer Aided Design (FMCAD). — 2019. — P. 212–220.

[14] Farzan Azadeh, Kincaid Zachary. Strategy Synthesis for Linear Arithmetic Games // Proc. ACM Program. Lang. — 2017. — Dec. — Vol. 2, no. POPL. — Access mode: https://doi.org/10.1145/3158149.

[15] Fedyukovich Grigory, Gupta Aarti. Functional Synthesis with Examples // Principles and Practice of Constraint Programming / Ed.

by Thomas Schiex, Simon de Givry. — Cham : Springer International Publishing, 2019. — P. 547–564.

[16] Fedyukovich Grigory, Gurfinkel Arie, Gupta Aarti. Lazy but Effective Functional Synthesis // Verification, Model Checking, and Abstract Interpretation. — 2019. — Jan. — Access mode: http://dx.doi.org/10.1007/978-3-030-11245-5_5.

[17] Fedyukovich Grigory, Gurfinkel Arie, Sharygina Natasha. Automated Discovery of Simulation Between Programs // LPAR. — Vol. 9450. — 2015. — P. 606–621.

[18] Functional Synthesis for Linear Arithmetic and Sets / Viktor Kuncak, Mikaël Mayer, Ruzica Piskac, Philippe Suter // Int. J. Softw. Tools Technol. Transf. — 2013. — Oct. — Vol. 15, no. 5–6. — P. 455–474. — Access mode: https://doi.org/10.1007/s10009-011-0217-7.

[19] Functional Synthesis via Input-Output Separation / Supratik Chakraborty, Dror Fried, Lucas M. Tabajara, Moshe Y. Vardi // FMCAD. — IEEE, 2018. — P. 1–9.

[20] Gulwani Sumit. Automating String Processing in Spreadsheets Using Input-output Examples // Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — POPL '11. — New York, NY, USA : ACM, 2011. — P. 317–330. — Access mode: http://doi.acm.org/10.1145/1926385.1926423.

[21] Gulwani Sumit, Polozov Alex, Singh Rishabh. Program Synthesis. — NOW, 2017. — August. — Vol. 4. — P. 1–119. — Access mode: https://www.microsoft.com/en-us/research/publication/program-synthesis/.

[22] Huang K. Qiu X. Wang Y. DRYADSYNTH: A Concolic SyGuS Solver. — Access mode: https://engineering.purdue.edu/~xqiu/DryadSynth.pdf (online; accessed: 10.09.2020).

[23] Jacobs Swen, Kuncak Viktor. Towards Complete Reasoning about Axiomatic Specifications // Verification, Model Checking, and Abstract Interpretation. — Springer, 2011. — Jan. — Access mode: http://dx.doi.org/10.1007/978-3-642-18275-4_20.

[24] Jha Susmit, Seshia Sanjit A. A theory of formal synthesis via inductive learning // Acta Informatica. — 2017. — Vol. 54. — P. 693–726.

[25] Komuravelli Anvesh, Gurfinkel Arie, Chaki Sagar. SMT-Based Model Checking for Recursive Programs // Computer Aided Verification / Ed. by Armin Biere, Roderick Bloem. — Cham : Springer International Publishing, 2014. — P. 17–34.

[26] Lezama A Solar. Program synthesis by sketching : Ph. D. thesis / A Solar Lezama ; Citeseer. — 2008.

[27] Rabe Markus N. Incremental Determinization for Quantifier Elimination and Functional Synthesis // CAV, Part II. — Vol. 11562. — Springer, 2019. — P. 84–94.

[28] Raghothaman Mukund, Reynolds Andrew, Udupa Abhishek. The SyGuS Language Standard Version 2.0. — 2019.

[29] Raghothaman Mukund, Udupa Abhishek. Language to Specify Syntax-Guided Synthesis Problems. — arXiv : cs.PL/http://arxiv.org/abs/1405.5590v2.

[30] Refutation-based synthesis in SMT / Andrew Reynolds, Viktor Kuncak, Cesare Tinelli et al. // Formal Methods in System Design. — 2017. — Feb. — P. 1. — Access mode: http://dx.doi.org/10.1007/s10703-017-0270-2.

[31] Reynolds Andrew. Conflicts, Models and Heuristics for Quantifier Instantiation in SMT // Vampire 2016. Proceedings of the 3rd Vampire Workshop / Ed. by Laura Kovacs, Andrei Voronkov. — Vol. 44 of EPiC Series in Computing. — EasyChair, 2017. — P. 1–15. — Access mode: https://easychair.org/publications/paper/8CX.

[32] Satisfiability modulo theories / Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, Cesare Tinelli // Handbook of Satisfiability. — 1 edition. — Frontiers in Artificial Intelligence and Applications no. 1. IOS Press, 2009. — Jan. — P. 825–885. — ISBN: 9781586039295.

[33] Search-based Program Synthesis / Rajeev Alur, Rishabh Singh, Dana Fisman, Armando Solar-Lezama // Communications of the ACM. — 2018. — 11. — Vol. 61. — P. 84–93.

[34] Shoham Sharon. Undecidability of Inferring Linear Integer Invariants. — 2018. — 1812.01069.

[35] Software Synthesis Procedures / Viktor Kuncak, Mikaël Mayer, Ruzica Piskac, Philippe Suter // Commun. ACM. — 2012. — Feb. — Vol. 55, no. 2. — P. 103–111. — Access mode: `http://doi.acm.org/10.1145/2076450.2076472`.

[36] SyGuS. SyGuS-Comp 2018 // SyGuS. — 2018. — Access mode: `http://sygus.org/competition.html` (online; accessed: 25.12.2018).

[37] SyGuS-Comp 2016: Results and Analysis / Rajeev Alur, Dana Fisman, Rishabh Singh, Armando Solar-Lezama // Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016 / Ed. by Ruzica Piskac, Rayna Dimitrova. — Vol. 229 of EPTCS. — 2016. — P. 178–202. — Access mode: `https://doi.org/10.4204/EPTCS.229.13`.

[38] SyGuS-Comp 2018: Results and Analysis / Rajeev Alur, Dana Fisman, Saswat Padhi et al. // CoRR. — 2019. — Vol. abs/1904.07146. — 1904.07146.

[39] Alur Rajeev, Bodik Rastislav, Juniwal Garvit et al. Syntax-Guided Synthesis. — 2013.

[40] Synthesizing Framework Models for Symbolic Execution / Jinseong Jeon, Xiaokang Qiu, Jonathan Fetter-Degges et al. // Proceedings of the 38th International Conference on Software

Engineering. — ICSE '16. — New York, NY, USA : ACM, 2016. — P. 156–167. — Access mode: http://doi.acm.org/10.1145/2884781.2884856.

[41] Katis Andreas, Fedyukovich Grigory, Guo Huajun et al. Validity-Guided Synthesis of Reactive Systems from Assume-Guarantee Contracts. — 2017. — 1709.04986.

[42] Caulfield Benjamin, Rabe Markus N., Seshia Sanjit A., Tripakis Stavros. What's Decidable about Syntax-Guided Synthesis? — 2015. — 1510.08393.

[43] de Moura Leonardo, Bjørner Nikolaj. Z3: An Efficient SMT Solver // Tools and Algorithms for the Construction and Analysis of Systems / Ed. by C. R. Ramakrishnan, Jakob Rehof. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. — P. 337–340.

[44] Герасимов А. С. Курс математической логики и теории вычислимости: Учебное пособие. — Издательство «Лань», 2014. — ISBN: 978-5-8114-1666-0.