

Санкт-Петербургский государственный университет

Кафедра системного программирования

Мухин Артем Михайлович

Синтез структур данных

Курсовая работа

Научный руководитель:
программист JetBrains Мордвинов Д. А.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Software Engineering Chair

Artem Mukhin

Data Structure Synthesis

Graduation Thesis

Scientific supervisor:
Dmitry Mordvinov

Reviewer:

Saint-Petersburg
2017

Оглавление

Введение	4
1. Постановка задачи	5
2. Обзор	6
2.1. Дедуктивный синтез	6
2.2. Индуктивный синтез	6
2.3. Синтез, управляемый оракулом	7
2.4. FlashFill	8
2.5. Синтез структур данных	8
3. Текущие результаты	9
Список литературы	11

Введение

Во многих программах используются специальные структуры данных, не содержащиеся явно в стандартных библиотеках языков программирования. Таковыми являются структуры, составленные из известных коллекций с помощью композиции или изменения некоторых свойств. Реализация таких структур — монотонная и сложная работа, в ходе которой зачастую возникают ошибки как на этапе подбора оптимальных структур (т.к. для этого программисту необходимо знать асимптотики различных операций над структурами данных), так и на этапе реализации. Поэтому программистам был бы полезен инструмент, позволяющий по короткой декларативной спецификации автоматически получить исходный код оптимальной и корректной реализации структуры данных.

Синтез программ — это задача автоматического поиска программы, написанной на заданном языке программирования и удовлетворяющей некоторым ограничениям, которые задает пользователь. В отличие от трансляторов, синтезаторы осуществляют поиск требуемой программы в пространстве всех возможных программ, а не просто переводят программу с одного языка на другой. Ограничения на требуемую программу могут быть спецификацией на некотором промежуточном языке, примерами ввода-вывода, описанием на естественном языке, частично написанными программами и т.д. Синтезатор структур данных — это инструмент, позволяющий получить реализацию (на заданном пользователем языке программирования) автоматически сконструированной структуры данных.

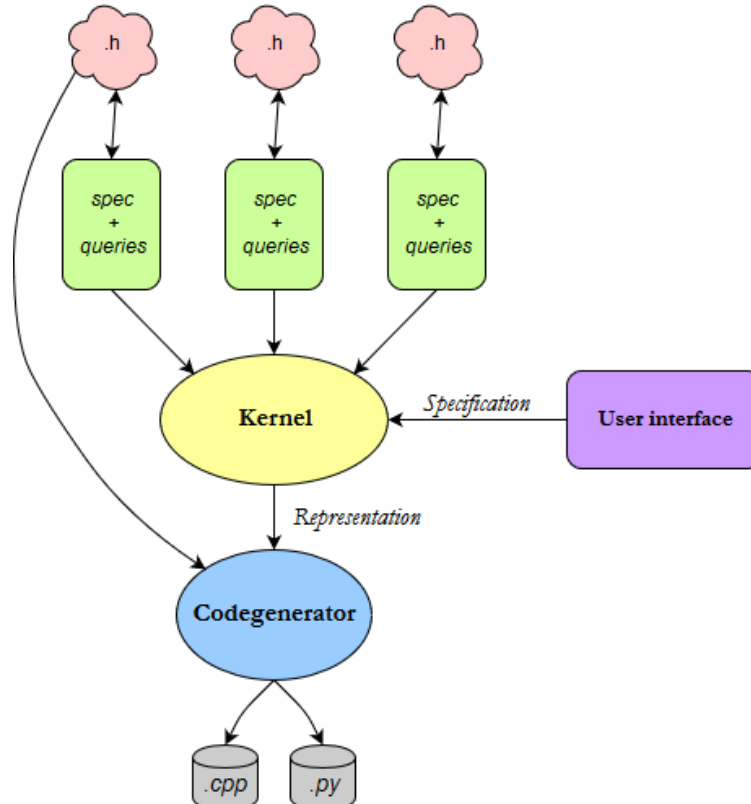
1. Постановка задачи

Целью данной работы является создание синтезатора структур данных, подходящего для более широкого класса задач, чем существующие. Для этого были поставлены следующие задачи:

- исследовать существующие подходы к синтезу программ;
- спроектировать декларативный язык спецификации структур данных, достаточно мощный для описания этих задач;
- разработать алгоритм синтеза структур данных;
- сравнить получившийся синтезатор с существующими.

По проекту ведется параллельная курсовая работа. Автор данной курсовой работы занимается созданием ядра синтезатора, а его коллега — реализацией структур данных и генерацией кода.

Рис. 1: Архитектура синтезатора



2. Обзор

Синтез программ является одной из наиболее значимых проблем информатики и искусственного интеллекта. Успешное решение этой задачи требует преодоления двух основных проблем:

1. необходимо предоставить пользователю достаточно простой и удобный способ спецификации желаемой программы, который не был бы слишком сложен для компьютера;
2. синтез программ — сложнейшая комбинаторная проблема; во избежание взрыва количества программ приходится как можно сильнее сокращать пространство перебора.

Исторически реализация синтезаторов программ стала возможной после появления инструментов для автоматического доказательства теорем.

2.1. Дедуктивный синтез

Первые подходы [3], [9] к синтезу программ были основаны на дедукции. Идея *дедуктивного синтеза* — автоматически доказать заданную пользователем спецификацию и затем использовать это доказательство для конструирования подходящей программы. Основной минус этого подхода заключается в том, что пользователю приходится составлять полную и формальную спецификацию, что во многих случаях не проще, чем написать требуемую программу вручную.

2.2. Индуктивный синтез

Для упрощения предварительной работы пользователя был придуман метод *индуктивного синтеза*. В этом подходе вместо полных спецификаций программы используются индуктивные ограничения, такие как примеры ввода-вывода и демонстрации работы программы. В 1970-е годы этот подход был реализован в нескольких инструментах для

синтеза небольших программ на языке LISP [7]. Сегодня для синтеза программ чаще применяют техники, сочетающие в себе и дедуктивный, и индуктивный подходы.

2.3. Синтез, управляемый оракулом

Одним из современных подходов является синтез на основе *частично написанных программ* (программ с «дырками»). В качестве спецификации желаемой программы выступает некоторый каркас, содержащий некоторые заглушки, на место которых синтезатор должен подставить сгенерированные им выражения языка, а также ограничения в виде *утверждений* (assertions), которым должна удовлетворять итоговая программа. Первой известной реализацией этого подхода является инструмент Sketch [8], созданный в 2008 году. Важнейшей находкой автора Sketch является следующее наблюдение: в отличие от синтеза, задачи второго порядка, верификация программ является задачей первого порядка и решается гораздо проще. При этом задача синтеза программы сводится к верификации программ, полученных генератором произвольных программ на фиксированном языке. Для каждого очередного кандидата требуется найти конструктивный контрпример, который в дальнейшем используется как дополнительное ограничение на искомую программу. Такой подход получил название *CEGIS* (counterexample-guided inductive synthesis).

Также стоит упомянуть инструмент для синтеза преобразований списков λ^2 [2]. Этот синтезатор принимает на вход несколько примеров преобразований списка или дерева и возвращает функцию, удовлетворяющую этим примерам. λ^2 получился достаточно мощным, чтобы синтезировать весьма нетривиальные операции, такие как декартово произведение списков и удаление дубликатов из списка. Данная работа интересна тем, что сочетает сразу несколько техник синтеза: индукцию для обобщения входных примеров; дедукцию для опровержения и генерации новых примеров; заполнение «дырок» в промежуточных программах-кандидатах.

2.4. FlashFill

На сегодняшний день одним из наиболее известных и успешных коммерческих продуктов, использующих синтез программ (а именно — индуктивный синтез на основе примеров), является Microsoft Excel с технологией FlashFill [4]. Эта технология позволяет автоматически синтезировать программу, выполняющую преобразования над данными в ячейках электронной таблицы, основываясь на нескольких примерах, выполненных пользователем вручную. Таким образом, пользователю достаточно заполнить несколько первых ячеек столбца, после чего FlashFill предложит заполнить остальные ячейки самостоятельно в соответствии с пользовательскими примерами.

2.5. Синтез структур данных

Перейдем к обзору решений задачи синтеза структур данных.

В 1970-е годы задача выбора оптимальной реализации структуры данных была рассмотрена во время проектирования языка SETL, ориентированного на работу с множествами [6]. Для улучшения производительности требовалось выбрать наиболее оптимальные для конкретных задач реализации структур данных множество и словарь.

Кроме того, были попытки синтезировать структуры данных с помощью реляционной алгебры (например, RelC [1]).

Работа, представляющая для нас наибольший интерес, — это синтезатор структур данных Cozu [5]. Авторы данной работы предложили подход к синтезу структур данных, основанный на идее CEGIS. С помощью Cozu работа программиста сводится к описанию требуемой ему структуры данных на языке спецификации высокого уровня, а поиск и реализацию подходящих структур данных берет на себя синтезатор. Cozu имеет довольно скудный язык спецификации и способен решать некоторые несложные задачи, а именно — фильтрацию элементов коллекции по предикатам, содержащим только имена переменных, полей и знаки сравнения.

3. Текущие результаты

На данный момент собрана некоторая коллекция задач, потенциально подходящих для синтеза, и спроектирован начальный вариант языка спецификации, позволяющий описать эти задачи.

Для сравнения рассмотрим грамматику языка спецификации Cozy.

$$\begin{aligned} \langle predicate \rangle ::= & \text{True} \mid \text{False} \\ & \mid \langle var \rangle \langle comparison \rangle \langle var \rangle \\ & \mid \langle predicate \rangle \text{And} \langle predicate \rangle \\ & \mid \langle predicate \rangle \text{Or} \langle predicate \rangle \\ & \mid \text{Not} \langle predicate \rangle \end{aligned}$$
$$\langle var \rangle ::= \langle field \rangle \mid \langle query-var \rangle$$
$$\langle comparison \rangle ::= '=' \mid '>' \mid '>=' \mid '<' \mid '<=' \mid '!='$$

Язык Cozy имеет несколько проблем.

Во-первых, на этом языке возможно описать только запросы-фильтрации (вернуть элементы коллекции, удовлетворяющие предикату). Мы хотели бы иметь язык, достаточно богатый для описания запросов более сложных видов. Поэтому язык должен поддерживать не только *filter*, но и, по крайней мере, *map* (т.к. многие задачи подразумевают не просто получение элементов коллекции, но и выполнение каких-либо действий над этими элементами).

Во-вторых, язык Cozy не поддерживает арифметику. Учитывая мощности современных SMT-решателей, добавление линейной арифметики выглядит возможным.

Для простоты мы не будем приводить здесь полную грамматику языка спецификации (кроме того, грамматика, скорее всего, будет изменяться в ходе разработки всего проекта). Вместо этого ограничимся кратким обзором основных возможностей языка.

В качестве наглядного примера рассмотрим задачу поиска инцидентных данной вершине дуг в графе, спецификация которой приведена в Листинге 1.

Listing 1: Поиск инцидентных данной вершине дуг в графе

```
record Edge { src: Int, dst: Int }  
graph: Collection<Edge>  
query findEdges(node: Int) =  
    filter (\edge -> edge.src == node || edge.dst == node) graph
```

Спецификация состоит из объявлений типов записей (элементов коллекций) и типов коллекций, а также из определений запросов к коллекциям. Синтаксис определения запросов во многом подобен синтаксису языка Haskell. Пользователь может использовать функции и комбинаторы из заранее заданного набора: *filter*, *map*, *fold*, *min*, *max*, *size*, *sort*.

Список литературы

- [1] Data Representation Synthesis / Peter Hawkins, Alex Aiken, Kathleen Fisher et al. // Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation. — PLDI '11. — New York, NY, USA : ACM, 2011. — P. 38–49. — URL: <http://doi.acm.org/10.1145/1993498.1993504>.
- [2] Feser John K., Chaudhuri Swarat, Dillig Isil. Synthesizing Data Structure Transformations from Input-output // SIGPLAN Not. — 2015. — Vol. 50, no. 6. — P. 229–239. — URL: <http://doi.acm.org/10.1145/2813885.2737977>.
- [3] Green C. Cordell. Application of theorem proving to problem solving // IJCAI. — 1969. — P. 219–240.
- [4] 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. — URL: <http://doi.acm.org/10.1145/1926385.1926423>.
- [5] Loncaric Calvin, Torlak Emina, Ernst Michael D. Fast synthesis of fast collections // PLDI 2016: Proceedings of the ACM SIGPLAN 2016 Conference on Programming Language Design and Implementation. — 2016. — P. 355–368.
- [6] Schwartz J. T. Automatic Data Structure Choice in a Language of Very High Level // Proceedings of the 2Nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. — POPL '75. — New York, NY, USA : ACM, 1975. — P. 36–40. — URL: <http://doi.acm.org/10.1145/512976.512981>.
- [7] Shaw David E., Swartout William R., Green C. Cordell. Inferring LISP Programs from Examples // Proceedings of the 4th International Joint Conference on Artificial Intelligence - Volume 1. — IJCAI'75. —

San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1975. — P. 260–267. — URL: <http://dl.acm.org/citation.cfm?id=1624626.1624666>.

- [8] Solar-Lezama Armando. Program Synthesis by Sketching : Ph.D. thesis / Armando Solar-Lezama. — Berkeley, CA, USA : University of California at Berkeley, 2008. — AAI3353225.
- [9] Zohar Manna Richard J. Waldinger. Toward automatic program synthesis // Commun. ACM. — 1971. — P. 151–165.