информатика

Компьютерные инструменты в образовании, 2022
№ 1: 16–29
УДК: 519.7:004.43
http://cte.eltech.ru
doi:10.32603/2071-2340-2022-1-16-29

РАЗРАБОТКА ПАКЕТА Python 3 ДЛЯ ВЫЧИСЛЕНИЙ В ТЕОРИИ МУЛЬТИОПЕРАЦИЙ

Еременко Д. А.[1], аспирант, ✉ er_92@list.ru

[1]Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина), 5, корп. 3, ул. Профессора Попова, 197376, Санкт-Петербург, Россия

Аннотация

В работе решается задача разработки инструментария, позволяющего проводить компьютерные вычисления для получения новых результатов в теории мультиопераций. В статье приведены различные методы представления операций и мультиопераций и описаны алгоритмы для вычисления суперпозиции операций и мультиопераций. Также в работе приводятся исследования различных структур языка Python 3 и поиск наиболее подходящих для реализации представлений операций и мультиопераций. Основываясь на полученных результатах исследований структур данных, была разработана и реализована архитектура пакета Python 3 для моделирования алгебр операций и мультиопераций в теории мультиопераций.

Ключевые слова: *операции, мультиоперации, Python 3, алгебры операций, алгебры мультиопераций.*

Цитирование: Еременко Д. А. Разработка пакета Python 3 для вычислений в теории мультиопераций // Компьютерные инструменты в образовании. 2022. № 1. С. 16–29. doi: 10.32603/2071-2340-2022-1-16-29

1. ВВЕДЕНИЕ

Теория мультиопераций относится к пограничным разделам информатики, дискретной математики и математической логики. Данный раздел математики изучает операции и мультиоперации, определенные на конечных множествах, и алгебры операций и мультиопераций. Такие функции используются в математической логике и в универсальной алгебре [1], в частности, в теории клонов [2–4]. Для решения различных задач в теории мультиопераций можно эффективно использовать компьютерные вычисления. Например, в работе [5] были найдены все минимальные алгебры бинарных операций ранга 3, в работе [6] были найдены все минимальные алгебры бинарных мультиопераций ранга 3, а в работе [7] были сформулированы теоремы о совершенной связи Галуа для унарных операций и мультиопераций различных рангов.

Однако несмотря на ведущиеся исследования в современных средствах компьютерной математики (таких как MATLAB, Maple, Mathematica), функционал для работы с алгебрами операций и мультиопераций отсутствует. Самым подходящим из существующих систем компьютерной алгебры можно считать Sage. Используя класс

sage.sets.finite_set_maps [8], можно смоделировать алгебры унарных операций различного ранга в сигнатуре <*> (суперпозиция операций), представив их в виде моноида с операцией суперпозиции. Но даже для изучения унарных операций и мультиопераций данного инструментария недостаточно. В **sage.sets.finite_set_maps** поддерживается только одна сигнатура, в то время как в теории мультиопераций алгебры рассматриваются в различных сигнатурах (например метаоперации пересечения, объединения или более сложные сигнатуры, включающие в себя несколько метаопераций).

В связи с этим была поставлена задача создать программный комплекс, который можно было бы использовать для вычислений и получения новых результатов в теории мультиопераций. В качестве языка программирования был выбран язык Python 3 как один из самых популярных и выразительных современных языков.

2. ВВОДИМЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Под n-местной операцией на множестве A понимают отображение из A^n в A.

Под рангом операции понимается мощность множества A. $k = |A|$.

Множество всех n-местных операций на A обозначим через P_A^n.

Операции $f \in P_A^n$, где $A = \{a_0, ..., a_{k-1}\}$ можно представить как отображения

$$f : \{2^0, ..., 2^{k-1}\}^n \mapsto \{1, ..., 2^{k-1}\},$$

получаемые из f при кодировке $a_i \mapsto 2^i$.

При этом операцию f зададим векторной формой $\{\alpha_0, ..., \alpha_{k^n-1}\}$, где $\alpha_i \in \{2^0, ..., 2^{k-1}\}$, $\alpha_i = f(2^{j_1}, ..., 2^{j_n})$, где $(j_1, ..., j_n)$ есть предствление i в системе исчисления по основанию k n-разрядным числом.

Более подробно с примерами построения векторных форм для операций можно ознакомиться в работе [5].

Под суперпозицией операций $f \in P_A^n$ и $f_1, ..., f_n \in P_A^m$ понимают:

$$\big(f * (f_1, ..., f_n)\big)(a_1, ..., a_m) = f\big(f_1(a_1, ..., a_m), ..., f_n(a_1, ..., a_m)\big).$$

Обобщением понятия операции является мультиоперация. Обобщение заключается в том, что образом отображения может являться не один элемент, а множество элементов из A.

Пусть $B(A)$ — множество всех подмножеств A. Отображение g декартовой степени A^n в $B(A)$ называется n-местной мультиоперацией на A и обозначается как

$$g : A^n \to B(A).$$

Если все образы — одноэлементные множества, то g будет являться n-местной операцией.

Множество всех n-местных мультиопераций на A обозначим через M_A^n.

Мультиоперации $g \in M_A^n$ на конечном множестве $A = \{a_0, ..., a_{k-1}\}$ можно представить как отображения $g : \{2^0, ..., 2^{k-1}\}^n \to \{0, 2^0, ..., 2^{k-1}\}$, получаемые при кодировке

$$a_i \to 2^i; \quad \emptyset \to 0; \quad \{a_{i_1}, ..., a_{i_s}\} \to 2^{i_1} + ... + 2^{i_s}.$$

Мультиоперацию g зададим векторной формой

$$(\alpha_0, ..., \alpha_{k^n-1}), \quad \text{где } \alpha_i \in \{2^0, ..., 2^{k-1}\}, \quad \alpha_i = f(2^{j_1}, ..., 2^{j_n}).$$

При этом $(j_1, ..., j_n)$ есть представление i в системе исчисления по основанию k n-разрядным числом.

Алгеброй мультиопераций размерности n ранга k называется любое $R \subseteq M_k^n$, содержащее все n-местные проекции, пустую мультиоперацию и замкнутые относительно метаопераций суперпозиции и разрешимости.

Обобщим понятие суперпозиции операции для мультиопераций.
Суперпозиция мультиопераций $g \in M_A^n$ и $g_1, ..., g_n \in M_A^m$:

$$(g * g_1, ..., g_n)(x_1, ..., x_m) = \bigcup_{b_i \in g_i(x_1, ..., x_m)} g(b_1, ..., b_n), \quad \text{где } x_i \in A.$$

Также для мультиопераций введена унарная метаоперация разрешимости:

$$(\mu_i g)(a_1, ..., a_n) = \{a | a_i \in g(a_1, ..., a_{i-1}, a, a_{i+1}, ..., a_n)\}.$$

3. МЕТОДЫ ПРЕДСТАВЛЕНИЯ ОПЕРАЦИЙ И МУЛЬТИОПЕРАЦИЙ

Для задания операций и мультиопераций будем использовать векторную форму, описанную ранее.

3.1. Представления с помощью массивов

Данное представление получается естественным образом из векторной записи операций при использовании кодировки исходных элементов множества $A : a_i \to i$. Мы будем представлять n-арную операцию ранга k в виде n-размерного массива размерности r. Отображение $A^n \to a_i | a_i \in A, A = \{0, 1, ..., k-1\}$ будем осуществлять с помощью индексации в массиве. Индексами будут значения аргументов операции, а значениями, хранящимися по соответствующим индексам, будут результаты операции от соответствующих аргументов.

Например операцию f в векторном виде (224122411) можно представить с помощью двухмерного массива размерности 3: $f_{arr} = \big[[1,1,2],[0,1,1],[2,0,0]\big]$.

Любой n-мерный массив можно представить с помощью одномерного массива, используя перекодировки индексов: $idx\_new = (idx_0, \ldots, idx_n)_k$ — представление n-разрядного числа по основанию k.

Таким образом, операция $f = (224122411)$ представима

— в векторной форме как (224122411);
— в виде двухмерного массива $f_{arr} = \big[[1,1,2],[0,1,1],[2,0,0]\big]$;
— в виде одномерного массива $f_{arr_f} = [1,1,2,0,1,1,2,0,0]$.

Пример перерасчета индексации:

$$f_{arr}[1][2] = f_{arr_f}\ [5], \text{ так как } (12_3 = 5).$$

Реализовать массивы в Python можно с помощью

— последовательностей — индексированных коллекций (**list**, **tuple**, **byte**, **bytearray**);
— отображения (**dict**);
— дополнительных типов (**array**).

Для представления мультиопераций есть два различных подхода. Первый заключается в том, чтобы представлять мультиоперации с помощью массивов, используя кодировку:

$$a_i \to 2^i; \quad \emptyset \to 0; \quad \{a_{i_1}, ..., a_{i_s}\} \to 2^{i_1} + ... + 2^{i_s}$$

— использование векторной записи представления мультиопераций. Второй подход заключается в том, чтобы представлять множества не суммой чисел, а вложенной в массив коллекцией Python.

Например мультиоперацию g в векторном виде (657124130) можно представить с помощью двухмерного массива размерности 3 с вложенными коллекциями, представляющими множества: $g_{arr} = \big[\big[\{1,2\},\{0,2\},\{0,1,2\}\big],\big[\{0\},\{1\},\{2\}\big],\big[\{0\},\{0,1\},\{\}\big]\big]$.

Двухмерный массив g_{arr} можно развернуть в одномерный:

$g_{arr_f} = \big[\{1,2\},\{0,2\},\{0,1,2\},\{0\},\{1\},\{2\},\{0\},\{0,1\},\{\}\big]$.

Для этого используется кодировка, такая же как и для операций.

Таким образом, мультиоперация g = (657124130) представима

— в виде одномерного массива
$g_{arr_{v1}} = [6,5,7,1,2,4,1,3,0]$;
— в виде двухмерного массива
$g_{arr_{v2}}\big[[6,5,7],[1,2,4],[1,3,0]\big]$;
— виде одномерного массива
$g_{arr_{s1}} = \big[\{1,2\},\{0,2\},\{0,1,2\},\{0\},\{1\},\{2\},\{0\},\{0,1\},\{\}\big]$;
— в виде двухмерного массива
$g_{arr_{s2}} = \big[\big[\{1,2\},\{0,2\},\{0,1,2\}\big],\big[\{0\},\{1\},\{2\}\big],\big[\{0\},\{0,1\},\{\}\big]\big]$.

Для выбора способа реализации массивов проведем исследование основных коллекций языка Python.

3.2. Методология исследования

Исследование коллекций сведём к следующим экспериментам для каждой коллекции:

1. Вычисление времени доступа к элементу массива (по индексу), представляющего собой унарные операции ранга 3.
2. Вычисление времени на создание массивов, представляющих собой унарные операции ранга 3.
3. Вычисление времени доступа к элементу массива (по индексу), представляющего собой бинарные операции ранга 3.
4. Вычисление времени на создание массивов, представляющих собой бинарные операции ранга 3.
5. Вычисление времени на создание массивов, представляющих собой унарные мультиоперации ранга 3.

В экспериментах будем проводить серию из пяти опытов для каждой коллекции, реализующей массив, а затем посчитаем среднее время. В каждом эксперименте будем измерять время выполнения кода с помощью функции **repeat** из модуля **timeit**. Для минимизации влияния кэширования на результаты экспериментов будем создавать коллекции с различными значениями, используя функцию **randint** из модуля **random**. В каждом опыте будем измерять время, затраченное на выполнение кода миллион раз.

3.3. Унарные операции ранга 3

Рассмотрим пункты 1, 2 исследования коллекций. Унарные операции ранга 3 представляются с помощью одномерных массивов размерности 3 с элементами, принимающими значения 0, 1, 2. Результаты 1 и 2 пунктов исследования представлены в таблице 1.

Таблица 1. Результаты 1, 2 пунктов исследования

creation	speed	title
0,0371888852	0,0178283874	**list**
0.0271082718	0,0187313538	**tuple**
0,1226309756	0,0197021542	**bytes**
0,1795118614	0,0238377390	**bytearray**
0,2408704746	0,0227110704	**array**
0,1078697254	0,0217435636	**dict**

Вывод: самые эффективные коллекции по операции индексации и по созданию коллекции — **list** и **tuple**. Их и будем рассматривать в дальнейшем.

3.4. Бинарные операции ранга 3

Рассмотрим пункты 3, 4 исследования коллекций. Бинарные операции ранга 3 представляются с помощью одномерных массивов размерности 9 либо с помощью двухмерных массивов размерности 3×3 с элементами 0, 1, 2. Для учета фактора перерасчета индексов при реализации бинарных операций одномерными массивами перед каждой операцией индексации добавлены операция умножения на число и сложение. Результаты 3 и 4 пункта исследований представлены в таблице 2.

Таблица 2. Результаты 3, 4 пунктов исследования

creation	speed	title
0,0849314958	0,0278363373	**tuple of tuple**
0,1302965494	0,0261544024	**list of lists**
0,049208932	0,0379425178	**tuple**
0,0631271186	0,0383025376	**list**

Вывод: из таблицы 2 следует, что одномерные массивы создаются в два раза быстрее, чем двухмерные; при этом индексация происходит медленнее. Таким образом, не удалось однозначно определить наилучший способ представления операций/мультиопераций.

3.5. Унарные мультиоперации ранга 3

Рассмотрим пункт 5 исследования структур данных. Унарные мультиоперации ранга 3 представляются либо в виде одномерного масива размерностью 3 с элементами [0–7] либо с помощью массива длиной 3×3 с вложенной коллекцией, представляющей собой множество, состоящее из элеметов 0, 1, 2. Для мультиопераций в качестве вложенных коллекций рассматриваются три варианта: **set**, **tuple**, **list**.

Время по доступу к элементам массивов, представляющих унарные мультиоперации, не отличается от времени доступа к аналогичным стуктурам для унарных операций. В связи с этим данный параметр не был измерен.

В таблице 3 приведено время на создание коллекций.

Таблица 3. Результаты 5 пункта исследования

creation	title
0,0848379532	**tuple of tuple**
0,4819560654	**tuple of set**
0,1303188006	**list of list**
0,4968571976	**list of set**
0,0371888852	**list**
0,0271082718	**tuple**

Вывод: несмотря на то, что массив с вложенной коллекцией **set** создается существенно медленней, чем массив с другими вложенными коллекциями, не стоит его исключать из потенциально рассматриваемых реализаций. При использовании вложенных коллекций **set** общее время может быть меньше, так как **set** имеет встроенный метод **update** и сохраняет уникальность коллекции, в то время как для **tuple** и **list** такого встроенного механизма нет. Необходимо рассмотреть различные варианты на практике.

Выводы по исследованию коллекций Python 3:

- Всего для реализации массивов подходят две коллекции: **list** и **tuple**.
- В качестве вложенных коллекций для представления мультиопераций могут быть использованы **set** и **list**.
- Необходимо оценить время для одномерных и двухмерных массивов, так как из эксперимента не очевидно, какое представление n-местных операций/мультиопераций будет эффективней.

4. РЕАЛИЗАЦИЯ МЕТАОПЕРАЦИЙ ДЛЯ ОПЕРАЦИЙ И МУЛЬТИОПЕРАЦИЙ

Ниже приведен алгоритм суперпозиции операций, представленных с помощью массивов.

Algorithm 1: Суперпозиция операций, представленных одномерными массивами

Data: r — ранг;
n — арность;
ops — операции, от которых вычисляется суперпозиция;
op — операция, для которой считается суперпозиция;
Result: O — операция, полученная в результате суперпозиции
id = 0;
foreach *row in table(ops)* **do**
    O[id] = op[to_decimal(to_str(row), base=r)];
    id = id+1;
end

Реализация представленного алгоритма в разработанном пакете.

```
def superposition_op(*ops, r):
*ops, op = ops
return tuple([op[int(''.join(str(s) for s in ss), r)]
for ss in zip(*ops)])
```

Для создания нового массива, реализованного с помощью **tuple**, необходимо передать конструктору класса **tuple** либо итерируемый объект, либо значения, разделенные запятыми. Массивы, реализованные с помощью **list**, создаются таким же образом. Кроме того, добавляется еще один способ — с помощью спискового включения.

Исследование сведём к следующим экспериментам для унарных и для бинарных операций, реализованных с помощью коллекций **list** и **tuple**:

1. Реализация суперпозиции с помощью генераторов.
2. Реализация суперпозиции с помощью передачи списка значений в конструктор коллекции.
3. Реализация суперпозиции с помощью механизма спискового включения.

Кроме того, для бинарных операций рассмотрим оба варианта, когда операции представляются двухмерными либо одномерными массивами. Методология измерения времени выполнения функций такая же, как и в предыдущем исследовании. Результаты измерения скорости выполнения суперпозиции операций представлены в таблице 4.

Таблица 4. Сравнение различных реализаций суперпозиции операций

unary	Binary 1d	Binary 2d	title
0,5721811936	1,2395862095	2,4041015	**List gen**
0,5861388534	1,2164262213	2,4753413	**Tuple gen**
0,3854481940	0,9410474606	1,6489777	**List comp**
0,2417752776	0,7084667883	1,0476483	**List from items**
0,1432888624	0,5980841064	0,7548499	**Tuple from items**

Вывод: реализация через одномерные массивы с помощью типа **tuple** самая эффективная, причем создание нового элемента происходит через передачу конструктору класса **tuple** значений, разделенных запятыми. Для реализации суперпозиции операций в общем виде лучше всего подходит механизм создания нового массива через механизм **list comprehension** (спискового включения).

Algorithm 2: Суперпозиция мультиопераций, представленных одномерными массивами

```
Data:
r — ранг;
n — арность;
dic_mop — словарь для кодирования мультиопераций;
mops — мультиоперации, от которых вычисляется суперпозиция;
mop — мультиоперация, для которой считается суперпозиция;
Result: M — мультиоперация, полученная в результате суперпозиции
id = 0;
foreach row in table(mops) do
    if 0 not in row then
        encoded_list = [];
        foreach num in row do
            encoded_list.append(dic_mop[num])
        end
        foreach code in cartesian_product(encoded_list) do
            M[id] = M[id] | mop[to_decimal(to_str(code), base=r)]
        end
    end
    id = id + 1
end
```

Реализация представленного алгоритма в разработанном пакете.

```
def superposition_mop(r, n, dic_mop, *mops):
rez = [0 for _ in range(r ** n)]
*mops, mop = mops
for idx, (ss) in enumerate(zip(*mops)):
if all((x != 0 for x in ss)):
l = [[i for i in dic_mop[x]] for x in ss]
for pairs in product(*l):
rez[idx] = rez[idx] | mop[int(''.join(str(s) for s in pairs), r)]
return tuple(rez)
```

При выборе способа представления мультиопераций воспользуемся выводами из предыдущего исследования. Для представления мультиопераций будем использовать только одномерные массивы, реализованные с помощью коллекции **tuple**. В качестве вложенных коллекций, представляющих собой множество, будем использовать **list** и **set**.

Таблица 5. Сравнение различных реализаций суперпозиции мультиопераций

unary	binary	title
0,577037845	4,3534857020	**1d tuples from items**
0,693794975	4,9760485730	**1d list from items**
0,923538479	5,9323611822	**tuple of set from items**
1,010826469	5,7935244474	**tuple of list from items**

Вывод: Самая эффективная реализация суперпозиции мультиопераций получается при использовании векторной формы записи мультиопераций и реализации через одномерные массивы с помощью коллекции **tuple**.

5. РАЗРАБОТКА БИБЛИОТЕКИ ДЛЯ ВЫЧИСЛЕНИЙ В ТЕОРИИ МУЛЬТИОПЕРАЦИЙ

5.1. Общая концепция реализации библиотеки

Для начала выделим объекты в теории мультиопераций, а далее их реализуем с помощью языка программирования Python. Вычисления в теории мультиопераций проводятся над объектами двух типов: операции / мультиоперации (представляются в виде массивов, реализованных с помощью типа **tuple**) и алгебры операций / мультиопераций (представляются с помощью объектов классов). Для представления алгебр используется объектно-ориентированный подход.

Классы алгебр задаются с помощью сигнатур, которые определяют тип (операции / мультиоперации, их арность и ранг), и метаоперации для вычисления алгебраических замыканий. Сами алгебры задаются с помощью порождающего множества — множества операций / мультиопераций, которые передаются конструктору класса.

Поскольку сигнатур бесконечное количество, то невозможно создать отдельный класс для каждой сигнатуры, поэтому классы алгебр создаются динамически в зависимости от сигнатуры. Для этого применяется порождающий паттерн — фабрика классов. Так как все классы реализуют один и тот же интерфейс, то при создании классов фабричной функцией используется механизм наследования от абстрактного класса **AlgBase**.

Методы динамически создаваемого класса связываются с реализацией с помощью соответствующих фабричных функций, которым передаются такие параметры как арность, ранг, тип (операции или мультиоперации), полученные из сигнатуры.

Таким образом, общую концепцию работы с пакетом можно свести к следующим пунктам:

— классы алгебры динамически генерируются с помощью фабричной функции на основе сигнатуры;
— алгебры представляются в виде объектов класса и задаются порождающим множеством;
— алгебраическое замыкание порождающего множества алгебры рассчитывается с помощью метода класса, и результат сохраняется в атрибуте алгебры в виде списка номеров операций, входящих в алгебраическое замыкание;
— список операций / мультиопераций, хранящихся в алгебре, получается с помощью итератора класса.

Перейдем к реализации описанной выше концепции.

5.2. Способ задания сигнатуры

Сигнатура задается с помощью строки параметров, разделенных запятой. Первый параметр — **op** / **mop** — алгебры операций или мультиопераций с заданием арности и ранга операций / мультиопераций через нижнее подчеркивание. Остальные параметры — символы метаопераций:

$*$ — суперпозиция;
-1 — разрешимость;
i — пересечение;
u — объединение.

Пример: **'op_2_3,*,-1'**

5.3. Описание интерфейса AlgBase

Опишем общий интерфейс, который должны реализовывать все классы алгебр.

1. Атрибут для хранения порождающего базиса, с помощью которого задается алгебра.
2. Атрибут для хранения списка операций, которые входят в алгебраическое замыкание.
3. Атрибут, отражающий, выполнено ли алгебраическое замыкание для данной алгебры.
4. Атрибут, отражающий, получено ли алгебраическое замыкание полностью, частично (если для функции, реализующей алгебраическое замыкания, был передан параметр, ограничивающий максимальное количество операций / мультиопераций в алгебре) или совпадает с множеством всех операций / мультиопераций.
5. Статический метод для кодирования мультиопераций числами.
6. Статический метод для декодирования чисел в мультиоперации.
7. Метод для вычисления алгебраического замыкания алгебры.
8. Метод для итерирования по алгебраическому замыканию алгебры.

Описанный интерфейс задается с помощью класса **AlgBase**, который представлен в модуле **Algebras** пакета **Algs**. На рис. 1 приведены атрибуты и методы интерфейса **AlgBase** из пакета **Algs**.

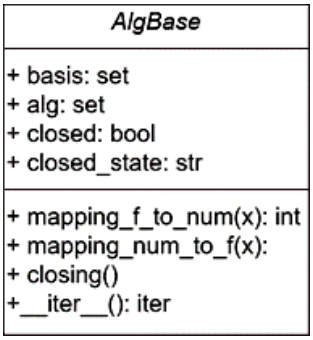

Рис. 1. Интерфейс **AlgBase**

5.4. Имплементация интерфейса AlgBase

Для имплементации интерфейса **AlgBase** в пакете **Algs** реализована функция **alg_factory**. В качестве обязательного параметра функции **alg_factory** передается сигнатура. Необязательные параметры: **parallel**, **bound**, **sheffer_list**, **chunk_size**.

В функции **alg_factory** происходит связывание методов динамически создаваемого класса **Alg** с их реализацией. Реализации методов получаются с помощью фабричных функций, которым передаются соответствующие параметры. Также в имплементацию **Alg** добавляется атрибут **metaoperations** — словарь, содержащий в качестве ключей символы метаопераций, а в качестве значений — функции, реализующие данные метаоперации. Помимо методов, объявленных в интерфейсе, при имплементации интерфейса добавляются вспомогательные методы. На рисунке 2 приведена имплементация интерфейса **AlgBase** с помощью класса **Alg**.

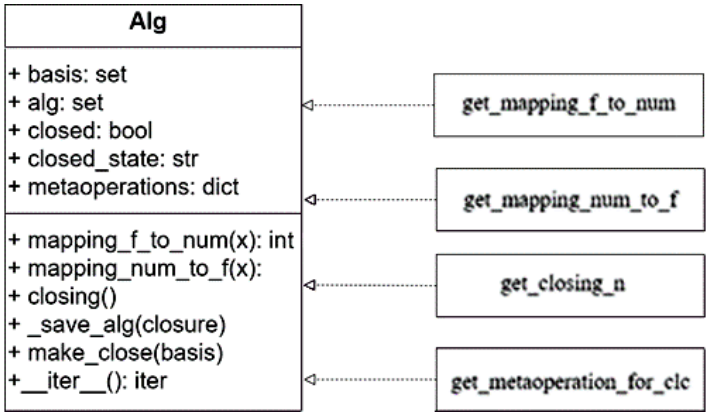

Рис. 2. Имплементация интерфейса

Атрибут **metaoperation** — словарь функций, реализующих метаоперации. Атрибут получается с помощью фабричной функции **get_mataopetation_for_clc**.

Метод **mapping_f_to_num** реализует кодирование мультиопераций числами. Имплементация метода **mapping_f_to_num** получается с помощью фабричной функции **get_mapping_f_to_num**.

Метод **mapping_num_to_f** реализует декодирование чисел в мультиоперации. Имплементация метода **mapping_num_to_f** получается с помощью фабричной функции **get_mapping_num_to_f** из пакета mappings.

Метод **closing** имплементируется с помощью двух вспомогательных методов: **save_alg** и **make_close**. Имплементация метода **make_close** получается с помощью фабричной функции **get_closing_n**. Стоит отметить, что реализация функции алгебраического замыкания (которая возвращается фабричной функции **get_closing_n**) настраивается с помощью дополнительных параметров, переданных функции **alg_factory**. Ниже перечислены необязательные параметры функции **alg_factory**, значения по умолчанию и их описание:

— **parallel (True)** — при значении True алгоритм алгебраического замыкания выполняется параллельно с помощью модуля multiprocessing и метода **apply_asinc**; при значении **False** алгоритм выполняется последовательно с помощью встроенной функции map;
— **chunk_size (10000)** — параметр, передающийся методу **apply_asinc**;
— **bound (False)** — параметр, задающий ограничение на размер алгебраического замыкания алгебры. Для задания используется число. Во время выполнения функции, реализующей вычисление алгебраического замыкания, на каждой итерации алгоритма считается количество операций / мультиопераций, входящих в алгебру. Алгоритм завершит работу, когда получено полное замыкание (то есть невозможно получить новую операцию / мультиоперацию) либо когда число операций / мультиопераций в алгебре достигает значения **bound**. При значении параметра **False** алгоритм завершит работу, только когда получит полное замыкание.
— **sheffer_list ()** — список операций / мультиопераций, которые порождают всё множество операций / мультиопераций. При выполнении функции алгебраического замыкания, если встретилась операция / мультиоперация, входящая в список, выполнение алгоритма прерывается, так как результат заведомо известен (всё множество операций / мультиопераций).

Реализация функции **alg_factory** находится в модуле **__init__ пакета Algs**.

5.5. Фабричные функции

При динамическом создании имплементации интерфейса связывание методов с их реализацией происходит с помощью фабричных функций. Данные функции на вход принимают параметры и возвращают функции, которые становятся методами для создаваемого класса. Все фабричные функции собраны в отдельные пакеты и объединены в пакете **Factories**. Существует четыре пакета с фабричными функциями:

— **mappings** — в пакете объединены функции, реализующие кодирование и декодирование операций/мультиопераций числами;
— **metaoperations** — в пакете объединены функции, реализующие все возможные метаоперации для операций и мультиопераций;
— **closing_by_signature** — в пакете объединены функции, реализующие алгоритм алгебраического замыкания (реализация зависит от сигнатуры алгебры);
— **args_generators** — в пакете объединены функции, реализующие генераторы для алгоритма алгебраического замыкания.

Архитектура разработанного пакета **Algs** представлена на рис. 3.

В данный момент разработанный пакет выложен в тестовом каталоге пакетов Python 3 и доступен для скачивания с помощью pip install -i https://test.pypi.org/simple/ algebras-of-multioperations.

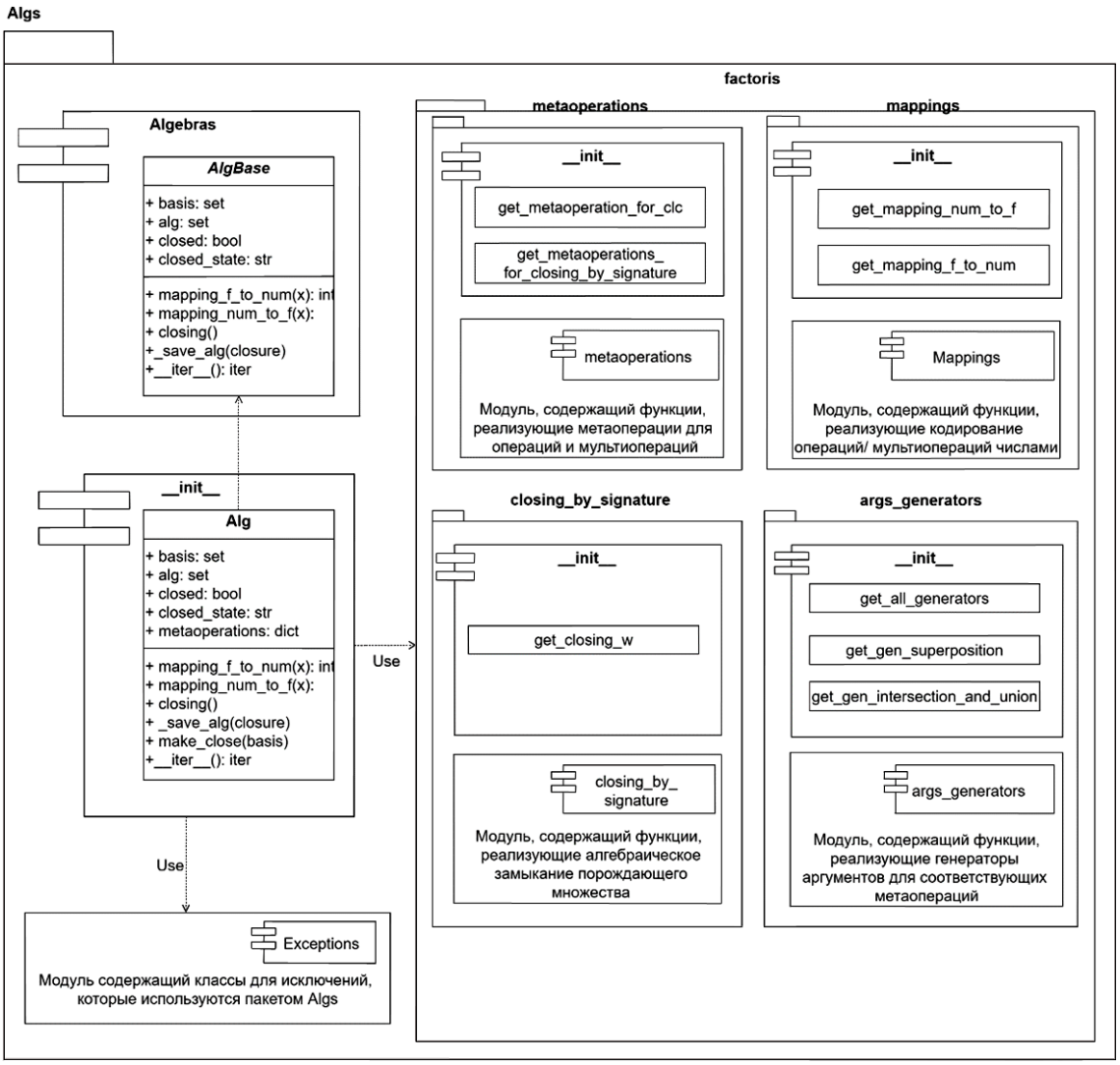

Рис. 3. Архитектура пакета **Algs**

В дальнейшем пакет будет расширяться новым функционалом, предназначенным для исследования теории мультиопераций.

6. ЗАКЛЮЧЕНИЕ

Разработанный пакет позволяет моделировать алгебры операций и мультиопераций различных арностей и рангов. На данный момент реализованы следующие метаоперации: суперпозиция, пересечение, объединение, операция разрешимости. Создание классов алгебр происходит динамически с помощью фабричных функций, что позволяет достаточно легко расширять функционал пакета новыми метаоперациями и алгоритмами. Так как язык Python 3 поддерживает расширение Python API (интерфейс прикладного программирования) для C и C++, то некоторые важные участки кода (например функция расчета алгебраического замыкания) в будущем могут быть перенесены на более быстрый язык программирования, например C, тем самым повысив быстродействие разработанного пакета.

Список литературы

1. *Пинус А. Г., Порошенко Е. Н., Судоплатов С. В.* Эрлагольская тетрадь. Избранные открытые вопросы по алгебре и теории моделей, поставленные участниками Эрлагольских школ-конференций // Новосибирск: Изд-во НГТУ 2018. С. 40.
2. *Behrisch, M., Vargas-García, E.* Unique inclusions of maximal C-clones in maximal clones // Algebra Universalis - P. 1–21.
3. *Бадмаев С. А.* Об одном максимальном клоне частичных ультрафункций на двухэлементном множестве // Журнал СФУ. Математика и физика. 2017. С. 140–145; doi: 10.17516/1997-1397-2017-10-2-140-145
4. *Pinus A. G.* Algebraically Equivalent Clones // Algebra Logic. 2017. Vol. 55, № 6. P. 501–506. doi: 10.1007/s10469-017-9420-2
5. *Еременко Д. А.* Минимальные алгебры бинарных операций ранга 3 // Компьютерные инструменты в образовании, 2021. № 4. С. 38–48.
6. *Тодиков С. И.* Алгоритм поиска минимальных алгебр бинарных мультиопераций ранга 3 // Современные информационные технологии и ИТ-образование. 2020. С. 841–850.
7. *Перязев Н. А.* Теория Галуа для конечных алгебр операций и мультиопераций ранга 2 // Известия Иркутского государственного университета. Серия Математика, 28 – 2019. С. 113–122.
8. *Sage 9.4 Reference Manual: Sets* Maps between finite sets [Electronic resource]. URL: https://doc.sagemath.org/html/en/reference/sets/sage/sets/finite_set_maps.html (дата обращения: 22.03.2022).

Поступила в редакцию 07.02.2022, окончательный вариант — 24.03.2022.

Еременко Дмитрий Александрович, аспирант кафедры ВТ факультета компьютерных технологий и информатики СПбГЭТУ «ЛЭТИ», ✉ **er_92@list.ru**

Computer tools in education, 2022
№ 1: 16–29
http://cte.eltech.ru
doi:10.32603/2071-2340-2022-1-16-29

Python 3 Package Developing for Computing in Theory of Multioperation

Eremenko D. A.[1], Postgraduate, ✉ er_92@list.ru

[1]Saint Petersburg Electrotechnical University,
5, building 3, st. Professora Popova, 197376, Saint Petersburg, Russia

Abstract

This work solves the problem of developing a toolkit that allows computer calculations to obtain new results in the theory of multioperations. The article presents various methods for representing operations and multioperations, and describes algorithms for calculating the superposition of operations and multioperations. Also, the work provides a study of various structures of the Python 3 and the search for the most suitable for the implementation of the representation of operations and multioperations. Based on the results of research on data structures, the architecture of the Python 3 package was developed and implemented for modeling algebras of operations and multioperations in the theory of multioperations.

Keywords: *operations, multioperations, Python 3, algebras of operations, algebras of multioperations.*

Citation: D. A. Eremenko, "Python 3 Package Developing for Computing in Theory of Multioperation," *Computer tools in education*, no. 1, pp. 16–29, 2022 (in Russian); doi: 10.32603/2071-2340-2022-1-16-29

References

1. A. G. Pinus, E. N. Poroshenko, and S. V. Sudoplatov, eds., *Erlagol notebook. Selected open-ended questions on algebra and model theory, posed by the participants of the Erlagol conference schools*, Novosibirsk, Russia: NSTU Publishing house, 2018 (in Russian).
2. M. Behrisch and E. Vargas-García, "Unique inclusions of maximal C-clones in maximal clones," *Algebra Universalis*, vol. 79, no. 31, pp. 1–21, 2018; doi: 10.1007/s00012-018-0497-9
3. S. A. Badmaev, "On Some Maximal Clone of Partial Ultrafunctions on a Two-element Set," *Siberian Federal University Journal. Mathematics & Physics*, no. 2, pp. 140–145, 2017; doi: 10.17516/1997-1397-2017-10-2-140-145
4. A. G. Pinus, "Algebraically Equivalent Clones," *Algebra Logic*, vol. 55, no. 6, pp. 501–506, 2017; doi: 10.1007/s10469-017-9420-2
5. D. A. Eremenko, "Minimal algebras of binary operations of rank 3," *Computer tools in education*, no. 1, pp. 38–48, 2020 (in Russian); doi: 10.32603/2071-2340-2020-1-38-48
6. S. I. Todikov, "Algorithm for Finding Minimal Algebras of Binary Multioperations of Rank 3," *Sovremennye informacionnye tehnologii i IT-obrazovanie [= Modern Information Technologies and ITEducation]* vol. 16, no. 4, pp. 841–850, 2020 (in Russian); doi: 10.25559/SITITO.16.202004.841-850
7. N. A. Peryazev, "Galois theory for finite algebras of operations and multioperations of rank 2," *Bulletin of Irkutsk State University. Series Mathematics*, vol. 28, pp. 113–122, 2019 (in Russian); doi: 10.26516/1997-7670.2019.28.113
8. Sage 9.4 Reference Manual: Sets»Maps between finite sets URL: https://doc.sagemath.org/html/en/reference/sets/sage/sets/finite_set_maps.html

Received 07-02-2022, the final version — 24-03-2022.

Dmitry Eremenko, Postgraduate, Department of Computer Science and Engineering, Faculty of Computer Science and Technology, Saint Petersburg Electrotechnical University, ✉ **er_92@list.ru**