

Оглавление

Глава 1. Введение и постановка задачи	2
1.1 Введение	2
1.2 Постановка задачи	2
Глава 2. Описание программы	3
2.1 Описание сущностей	3
2.1.1 Tuple	3
2.1.2 Function	4
2.1.3 Permutation	4
2.1.4 Predicate	4
2.2 Описание алгоритмов нахождения и перебора предикатов	4
2.2.1 PredicateFactory_P	4
2.2.2 PredicateFactory_O	5
2.2.3 PredicateFactory_E	5
2.2.4 PredicateFactory_L	5
2.2.5 PredicateFactory_C	6
2.2.6 PredicateFactory_B	6
2.3 Получение результатов	7
Глава 3. Результаты и заключение	8
Список литературы	9

Глава 1

Введение и постановка задачи

1.1 Введение

Как известно, каждый предполный класс в P_k можно задать как множество функций, сохраняющих некоторый предикат. Более того, все эти предикаты можно разделить на 6 попарно непересекающихся семейств: P, O, L, E, C, B , описание которых можно найти в книге Марченко С.С. «Функциональные системы с операцией суперпозиции».

1.2 Постановка задачи

- Найти все предикаты, описывающие предполные классы в P_4 , разбив их на семейства.
- Для каждой функции одной переменной из P_4 определить, каким предполным классам она принадлежит.

Глава 2

Описание программы

Для решения задачи была написана программа на языке программирования Java, исходный код которой можно найти, перейдя по следующей ссылке <https://github.com/zloi-timur/predicates>. Там же можно найти результаты и вспомогательные материалы, в том числе и этот текст.

Для создания проекта использовалась методология DDD (Domain-driven design), по этому хотелось бы начать с описания основных сущностей (реализованных в Java классах).

Т.к. при решении задачи часто приходилось перебирать однотипные объекты, многие классы реализуют интерфейсы `Iterator` и `Iterable`, что позволяет облегчить понимание исходного кода. При написании программы часто использовались следующие обозначения: *Dim* – “значность”, *Capacity* – «местность».

2.1 Описание сущностей

Все определения сущностей находятся в пакете `predicates.domain`

2.1.1 Tuple

Класс, который описывает упорядоченный набор чисел из `capacity` чисел от 0 до $dim - 1$. реализует интерфейсы `Iterator` и `Iterable`, что позволяет перебирать кортежи по порядку (лексикографическому).

2.1.2 Function

Класс, который описывает функцию, зависящую от *capacity* переменных чисел, из *Pdim*, реализует интерфейсы `Iterator` и `Iterable`, что позволяет перебирать функции по порядку столбцов значений (лексикографическому).

2.1.3 Permutation

Класс, который описывает перестановку чисел от 0 до *capacity* − 1. реализует интерфейсы `Iterator` и `Iterable`, что позволяет перебирать перестановки по порядку (лексикографическому).

2.1.4 Predicate

Описывает предикат, храня `Set<ImmutableList<Integer>` - множество векторов, удовлетворяющих предикату.

2.2 Описание алгоритмов нахождения и перебора предикатов

Для каждого семейства предикатов был написан класс `PredicateFactory_X`, где X – название соответствующего семейства. Каждый из них реализует интерфейсы `Iterable<Predicate>`, `Iterator<Predicate>`.

Все они лежат в пакете `predicates.factory`.

Т.к. при нахождении необходимых предикатов во многих случаях требуется перебор нескольких параметров и при некоторых комбинациях могут получаться одинаковые результаты, необходимо было не выдавать уже полученные идентичные предикаты. Эта проблема чаще всего решалась использованием структур данных, реализующих интерфейс `Set<Predicate>`.

Перейдем к подробному описанию каждого класса.

2.2.1 PredicateFactory_P

Каждый из предикатов класса задается перестановкой, которая является произведением циклов одной и той же простой длины (в нашем случае получается 2 цикла длины 2), по этому для нахождения всех предикатов

программа перебирает все перестановки, проверяет на выполнение вышеописанного условия (в классе `Permutation` есть соответствующий метод) и, при успешном результате проверки, строит предикат.

2.2.2 PredicateFactory_O

Семейство содержит любой двуместный предикат, который задает на E_k частичный порядок с наименьшим и наибольшим элементами (ограниченный частичный порядок).

Для нахождения всех таких предикатов программа перебирает все перестановки чисел от 0 до $dim - 1$ (от 0 до 3 в нашем случае), считая, что первый элемент перестановки будет наибольшим, последний — наименьшим.

Далее берутся все пары элементов (a_i, a_j) , где $i < j$ и перебираются все из $2^{(k-2)*(k-3)}$ вариантов считать или не считать, что эта пара упорядочена, причем отсекаются те конфигурации, в которых не выполняется транзитивность. Для каждого подошедшего варианта строится предикат.

2.2.3 PredicateFactory_E

Семейство состоит из всех двуместных предикатов, которые представляют собой отношения эквивалентности на E_k , отличные от полного и единичного отношений (нетривиальные отношения эквивалентности). Таким образом, каждое отношение эквивалентности из разбиает множество k на l классов попарно эквивалентных элементов, где $1 < l < k$.

Нахождение всех таких предикатов осуществляется при помощи перебора всех возможных разбиений и построения соответствующих предикатов.

2.2.4 PredicateFactory_L

Предикаты этого семейства существуют только при $k = p^l$ где p — простое число и $l > 0$. В этом случае на множестве E_k можно определить бинарную коммутативную операцию $+$ так, что $G = (E_k; +)$ будет являться абелевой группой периода p . Иными словами, в абелевой группе G порядок любого элемента, отличного от нуля группы, равен p .

Итак, если $k = p^l$ и $G = (E_k; +)$ — абелева группа периода p , то семейству L принадлежит предикат $x_1 + x_2 = x_3 + x_4$.

В нашем случае $p = l = 2$ и существует только одна таблица сложения, удовлетворяющая описанным условиям: 0, 1, 2, 3; 1, 0, 3, 2; 2, 3, 0, 1; 3, 2, 1, 0.

Для нахождения всех таких предикатов программа перебирает все перестановки чисел от 0 до $dim - 1$ (от 0 до 3 в нашем случае) и строит предикат $x_1 + x_2 = x_3 + x_4$. При $k = 4$ для всех перестановок предикаты оказались равными.

2.2.5 PredicateFactory_C

Семейство состоит из всех центральных предикатов.

Определение 1. Предикат (x_1, \dots, x_m) называется центральным, если он вполне рефлексивен, вполне симметричен, отличен от тождественно истинного предиката и существует такое непустое подмножество множества E_k [центр предиката], что предикату удовлетворяет всякий набор (a_1, \dots, a_m) из m_k , как только $x_1, \dots, x_m \cap \neq \emptyset$.

Определение 2. Предикат $p(x_1, \dots, x_m)$ называется вполне рефлексивным, если либо $m = 1$, либо если $m > 1$ и $p(a_1, \dots, a_m) = True$ для любого набора (a_1, \dots, a_m) из m_k , содержащего не более $m-1$ различных значений.

Определение 3. Предикат называется вполне симметричным, если он не меняется при любой перестановке переменных.

Для перебора всех предикатов нам необходимо перебрать размерность предиката.

Далее, для каждой размерности, мы строим минимальный вполне рефлексивный предикат. На следующем шаге мы перебираем все возможные центры и добавляем вектора, которые имеют с ним непустое пересечение, в предикат.

На последнем этапе перебора мы всеми возможными способами пытаемся его расширить еще не вошедшими векторами, учитывая условия симметричности и нетривиальности.

2.2.6 PredicateFactory_B

Если $h \geqslant , l \geqslant 1, k \geqslant h^l$ и q — отображение множества k на множество h^l , то семейству принадлежит предикат, который является полным прообразом l -й декартовой степени предиката τ при отображении q .

При $k = 4$ нам подходит $h = 3, 4$ и $l = 1$.

Для каждого из двух вариантов построим множества предикатов следующим образом:

1. переберем все отображения множества k на множество h^l .
2. для каждого отображения будем строить предикат `answer`, перебирая все возможные кортежи и проверяя, должны ли они входить в него при условии того, что `answer` должен быть прообразом предиката τ .

2.3 Получение результатов

Теперь, получив все искомые предикаты, необходимо проверить, какие функции сохраняют их. Для такой проверки в проекте есть класс `PredicateService`, который содержит метод с сигнатурой `public static boolean checkSave(Predicate predicate, Function function)`.

Теперь достаточно перебрать все пары, состоящие из функций и найденных предикатов, и применить к ним данный метод.

Глава 3

Результаты и заключение

1. Были найдены все предикаты, описывающие предполные классы в P_4 , и построена таблица(см. приложение А) принадлежности функций одной переменной четырехзначной логики этим предикатам.
2. Была создан Java-проект с архитектурой, позволяющей использовать его для задач, связанных с предикатами и предполными классами при значениях $k > 4$ (сейчас только при нахождении предикатов семейств L и B есть ограничения на k).

Список литературы

1. Марченкова С.С. Функциональные системы с операцией суперпозиции

-2	0	0
0	-3	0
0	0	-6

Рис. 3.1: Выигрыши первого игрока