Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б15-мм

Разработка транслятора подмножества языка Datalog в операции над многомерными матрицами

Лановая Александра Юрьевна

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н., С. В. Григорьев

Санкт-Петербург
2023

Оглавление

Введение **3**

1. Постановка задачи **5**

2. Обзор **6**

2.1. Анализ литературы в контексте Datalog 6

2.2. Определение и особенности многомерных матриц 8

2.3. Современные трансляторы и оптимизация запросов . . . 9

2.4. Вывод 10

3. Теория **11**

3.1. Многомерные матрицы 11

3.2. Транслятор 16

4. Реализация **22**

4.1. Представление матриц и библиотека операций над многомерными матрицами 22

4.2. Транслятор 23

4.3. Интерпретатор Datalog для проверки эквивалентности . 23

Заключение **25**

Список литературы **27**

Введение

В современном мире компьютерных технологий важность эффективной обработки данных невозможно переоценить. Datalog, являясь логическим языком запросов, обладает как структурной простотой, так и выразительностью, что делает его востребованным для работы с базами данных. Однако по мере увеличения объёмов и сложности данных, встаёт вопрос о расширении его возможностей через добавление новых конструкций. Важно отметить, что под расширением возможностей подразумевается не только улучшение производительности, но и обогащение самого языка, что позволяет эффективнее решать задачи обработки данных.

Datalog является языком запросов, основанным на реляционной алгебре и логическом программировании. Запросы в Datalog строятся с помощью правил, выражающих логические связи между отношениями в базе данных. Такие отношения обычно определяются через набор кортежей, или записей, каждая из которых описывает связь между элементами данных. В обычных реляционных базах данных эти отношения представлены в виде таблиц, где строки соответствуют кортежам, а столбцы – атрибутам этих отношений.

Особенность Datalog в том, что он позволяет работать не только с фактически хранящимися в базе данных отношениями, но и с производными (выводимыми) отношениями, которые формируются на базе существующих отношений при выполнении запросов. Это даёт возможность представлять более сложные и многоуровневые связи между данными, которые традиционно сложно выразить в стандартных реляционных моделях.

Известно, что бинарные отношения часто используются для моделирования простых зависимостей между объектами. Их задаются в виде разреженных двумерных булевых матриц [10] для удобства хранения и обработки. Однако, для эффективной обработки более сложных взаимосвязей, особенно между несколькими сущностями, или для оптимизации использования памяти и времени обращения, возникает потреб-

ность в обобщении этого подхода. Многоместные отношения представляют собой такое обобщение и могут быть представлены в виде многомерных разреженных булевых матриц. Это представление обладает структурной простотой, которая способствует не только наглядности данных, но и упрощает выполнение операций над ними. Применение методов линейной алгебры к таким структурам данных открывает дополнительные возможности для оптимизации, в том числе через параллельную обработку данных, что может существенно ускорить выполнение запросов и обработку больших объемов информации.

Несмотря на значительные исследования в области языков запросов и обработки данных, существует зазор в знаниях и практической реализации, особое внимание требуется уделить разработке трансляторов для Datalog, которые способны генерировать оптимизированный код для эффективной работы с многомерными матричными структурами. Неопределенность касается также оптимальных алгоритмов трансляции, которые должны учитывать специфику операций над многомерными матрицами и их применение в конкретных областях.

Стоит уделить внимание специфическим требованиям, которые необходимо учитывать при разработке транслятора для Datalog. Например, специфика может заключаться в необходимости оптимизации рекурсивных запросов, что является одной из ключевых особенностей Datalog, в разработке стратегий для эффективного управления памятью при работе с многомерными матрицами. Такие требования влияют на выбор алгоритмов и структур данных, которые должен поддерживать транслятор, для обеспечения высокой производительности и корректности преобразования запросов.

Таким образом, ключевой задачей является разработка и реализация транслятора для Datalog, который будет генерировать код, эффективно работающий с многомерными матричными структурами. Этот транслятор должен не только учесть специфические требования к хранению и обработке многоместных отношений, но и обеспечить высокую производительность операций, присущих логическим выводам в Datalog.

1. Постановка задачи

Целью работы является разработка транслятора языка Datalog в операции над многомерными матрицами, изучение его ограничений и доказательство корректности.

Для достижения данной цели были поставлены следующие задачи.

- Написать библиотеку для операций над многомерными матрицами, которая будет обеспечивать хранение и выполнение различных операций;
- Разработать транслятор, который будет отвечать за преобразование Datalog-запросов и правил в операции над многомерными матрицами, учитывая специфические требования языка и оптимизации производительности, а также обеспечить доказательство его корректности, подтверждая, что каждое преобразование соответствует семантике исходных запросов Datalog.
- Написать интерпретатор для Datalog, который будет использоваться для сравнения результатов вычислений и выполнения матричных операций после трансляции, с целью проверки корректности работы транслятора;
- Провести тестирование разработанных компонентов, включая библиотеку для операций над многомерными матрицами, транслятор и интерпретатор, для проверки их работы и эффективности;

2. Обзор

2.1. Анализ литературы в контексте Datalog

Для полноценного понимания возможностей и направлений развития языка Datalog представим более глубинный взгляд на литературные источники. Разработанный в 70-х годах язык Datalog получил новое дыхание с расширением области баз данных и искусственного интеллекта. В работе [3] освещаются первоначальные способности Datalog, представляющий собой язык для запросов к реляционным базам данных. Со временем авторы, такие как Ross (1990) [14] и Green (2007) [8], расширили его функциональность, включив рекурсивные запросы и стратегии оптимизации (способы написания запросов), что значительно увеличило его выразительную мощь и эффективность в вычислительном плане.

Рассмотрим синтаксис Datalog [2]. Пусть Pr – множество предикатов, $arity : Pr \to \mathbb{N}$ – функция арности, Cs – множество констант, V – множество переменных.

- **Term** – переменная или константа.

  $t ::= x \mid c$, где $x \in V,\ c \in Cs$

- Пусть $p \in Pr$ – предикат и $\bar{t}$ – последовательность термов, $|\bar{t}| = arity(p)$. **Atom** A – выражение вида:

  $A ::= p(\bar{t})$

  Термы $t_1, \cdots, t_n$ являются аргументами атома A.

- **Clause** C определяется так:

  $C ::= A_0 \leftarrow A_1, \cdots, A_m$

  Атом A_0 называется головой C, а список атомов $A_1, \cdots, A_m$ – его телом. Переименуем для удобства: H – голова, $B_1, \cdots B_m$ – тело.

- Когда $m = 0$, то есть $C \equiv H$, этот Clause представляет собой **Fact**. В случае, когда $m \geq 1$, Clause называется **Rule**.

- **Program** P – конечное множество Clauses.

  $P ::= C_0, \cdots, C_k$, где запятые означают конъюнкцию.

Рассмотрим семантику Datalog. Семантика неподвижной точки (fixpoint semantics) в контексте Datalog формализуется через последовательное применение оператора неподвижной точки к базе данных до достижения состояния, в котором дальнейшее применение оператора не ведет к изменению базы данных. Предположим, у нас есть Datalog программа с множеством правил и фактов.

Для формализации используем следующий подход:

- Пусть DB_0 – начальное состояние базы данных, содержащее все факты, известные до начала работы Datalog программы.

- Определим T_P как оператор, который применяет правила программы P к базе данных. Применение T_P к базе данных DB_i приводит к формированию нового множества фактов DB_{i+1}.

  $T_P : 2^P \to 2^P$.

- Формально, оператор T_P можно описать как функцию, которая из данного набора интерпретаций (отображений переменных в объекты) возвращает расширенный набор интерпретаций, включающий новые факты, полученные из применения правил программы: $T_P(DB_i) = DB_i \cup \{head(r) \mid r \subset P,\ body(r) \subseteq DB_i\}$

  Здесь $head(r)$ обозначает голову правила r, а $body(r)$ — тело правила r. Применение происходит для всех правил r программы P, где все факты в теле правила $body(r)$ удовлетворяются данными в DB_i.

- Фиксационную семантику F_P можно тогда определить как наименьшую неподвижную точку оператора T_P, такую, что:

  $F_P = lfp(T_P) = \bigcup_{i=0}^{\infty} DB_i$, $\theta_{lpf} = lfp(T_P)$, где $T_P(\theta_{lpf}) = \theta_{lpf}$, а lfp означает наименьшую неподвижную точку.

- Процесс нахождения фиксированной точки можно представить итеративно:

  1. $DB_{i+1} = T_P(DB_i)$
  2. Если $DB_{i+1} = DB_i$, тогда $F_P = DB_i$ и процесс останавливается.

  Этот процесс повторяется до тех пор, пока не будет достигнуто состояние, когда дальнейшее применение T_P не приводит к получению новых фактов, что означает достижение фиксированной точки. Полученный набор фактов и будет ответом на запрос.

Datalog находит применение в разнообразных областях. Исследование [7] демонстрирует использование Datalog в системах управления знаниями, где логическое программирование обеспечивает высокую степень абстракции для представления и манипуляции знаниями. В контексте программной инженерии, как показывают [17], Datalog используется для анализа программного кода, обеспечивая высокую точность и производительность при построении статических анализов. Применение Datalog в кибербезопасности освещается в работах [6], где он служит инструментом для анализа протоколов сетевого взаимодействия и выявления потенциальных уязвимостей.

2.2. Определение и особенности многомерных матриц

Прежде чем приступить к исследованию использования многомерных матриц [16] в базах данных, необходимо ввести их математическое определение и основные принципы работы. Многомерные матрицы представляет собой обобщение двумерного массива данных на случай с несколькими измерениями. Они могут эффективно хранить и представлять данные с несколькими атрибутами и стать ключевым инструментом для сложных аналитических запросов, благодаря своей способности к управлению большими объёмами данных и выполнению многомерных операций вычисления.

В литературе [5] подчеркивается ценность многомерных матриц для баз данных в больших системах сбора данных, где аналитические и вычислительные задачи требуют компактного представления подразумевающего множество связей между элементами данных. Эффективное использование таких структур представлено в работе [15], где исследуются способы сжатия, индексации и быстрого доступа к многомерным матрицам.

2.3. Современные трансляторы и оптимизация запросов

В последние годы был достигнут значительный прогресс в разработке трансляторов для языков запросов, включая Datalog. Современные трансляторы, такие как Soufflé [11] и LogicBlox [9], используют современные техники оптимизации для повышения эффективности выполнения запросов. Основными направлениями оптимизации являются мемоизация промежуточных результатов, использование индексов для ускорения доступа к данным и параллелизация вычислений. Такие подходы позволяют значительно ускорить процесс обработки данных и выполнения запросов, даже при работе с большими объемами информации.

Исследования, наподобие работы [4], подчеркивают оптимизацию запросов через различные алгебраические переформулировки и использование статистических моделей для эффективного выбора планов запросов. Применение машинного обучения в этой области, освещенное в работах [13], позволяет автоматизировать и уточнять процессы оптимизации запросов на основе сбора и анализа больших объемов производительностных данных.

Исследование аналогов трансляторов для Datalog показывает, что, несмотря на множество существующих реализаций, вопросы применения многомерных матриц в логическом программировании все еще не исследованы должным образом. Тем не менее, интеграция линейной алгебры в логические системы, как показано в работах [15], представляет собой перспективное направление. Многомерные матрицы могут играть

ключевую роль в оптимизации запросов, поскольку они позволяют выполнять многомерные операции над данными с высокой степенью параллелизма и эффективностью. Такие системы, как D4 [12] и Datafun [1], исследуют и разрабатываются с целью улучшения обработки запросов и открытия новых возможностей в области анализа данных.

2.4. Вывод

На основе проведенного анализа существующих работ можно сделать вывод о значимости разработки эффективных трансляторов для языка Datalog. Важно подчеркнуть, что вопросы эффективной обработки запросов в контексте баз данных и выполнения операций с многомерными структурами данных остаются высокоактуальными, и существующие методы их оптимизации продолжают развиваться. В связи с этим, разработка нового транслятора для Datalog представляет собой важную задачу, способную внести значительный вклад в развитие области обработки запросов к базам данных и открыть новые возможности для оптимизации и улучшения производительности логических выводов.

3. Теория

Перед тем как перейти к изложению реализации, стоит подчеркнуть ключевую задачу разработки: создание транслятора для языка Datalog, который способен на обработку запросов, используя многомерные матричные структуры. Важной частью проекта является выработка оптимальной схемы представления данных, которая позволит максимально использовать потенциал линейной алгебры для ускорения и упрощения процесса логического вывода. Таким образом, фокус реализации смещается в сторону эффективной интеграции классических операций над матрицами с семантикой Datalog. В этом контексте особое внимание уделяется методам представления и манипулирования многомерными разреженными булевыми матрицами, которые являются ключевыми для моделирования многоместных отношений в базах данных.

3.1. Многомерные матрицы

3.1.1. Многомерные матрицы и многоместные отношения

Известно, что при умножение двухмерных матриц, каждая из которых задает бинарное отношение, результирующая матрица будет задавать композицию изначальных отношений. Пусть $R \subseteq A \times B$ и $S \subseteq B \times C$, X – булева матрица R, Y – булева матрица S. Тогда матрица $R \circ S$ получается умножением X на Y.

Перейдем к случаю многомерных матриц и многоместных отношений. Композиция двух многоместных отношений определяется как операция, которая объединяет два отношения в одно новое отношение. Пусть у нас есть два многоместных отношения $\mathcal{R} \subseteq A_1 \times A_2 \times \cdots \times A_m$ и $\mathcal{S} \subseteq B_1 \times B_2 \times \cdots \times B_n$, где $A_1, A_2, \cdots, A_m$ и $B_1, B_2, \cdots, B_n$ – конечные непустые множества. R и S – булевы многомерные матрицы, которые задают многоместные отношения $\mathcal{R}$ и $\mathcal{S}$, соответсвенно. Докажем, что $(i, \cdots j, \cdots k, \cdots q) \in \mathcal{R} \circ \mathcal{S} \iff [rs]_{i \cdots j \cdots k \cdots q} = 1$, где $[rs]_{i \cdots j \cdots k \cdots q}$ – элемент результирующей матрицы RS.
$\Leftarrow$:$\sqsupset$ $\exists i, \cdots j, \cdots k, \cdots q : [rs]_{i \cdots j \cdots k \cdots q} = 1$, пусть умножение матриц про-

изводится по j и k измерениям, $j < k$, тогда по определению $\vee : \exists x$ для которого: $\bigvee_{x=1}^{n}[r]_{i\cdots x\cdots k\cdots q} \wedge [s]_{i\cdots j\cdots x\cdots q} = 1$, где n представляет собой количество элементов в k-ом измерении, умножаемое на первую многомерную матрицу R, или количество элементов в j-ом измерении, умножаемое на вторую многомерную матрицу S. Из чего следует, что $(i, \cdots x, \cdots k, \cdots q) \in \mathcal{R}$ и $(i, \cdots j, \cdots x, \cdots q) \in \mathcal{S}$. Тогда получаем по определению композиции отношений, что $(i, \cdots j, \cdots k, \cdots q) \in \mathcal{R} \circ \mathcal{S}$
$\Rightarrow: \sqsupset \exists i, \cdots j, \cdots k, \cdots q : (i, \cdots j, \cdots k, \cdots q) \in \mathcal{R} \circ \mathcal{S}$, тогда по определению композиции отношений $\exists x : (i, \cdots x, \cdots k, \cdots q) \in \mathcal{R}$ и $(i, \cdots j, \cdots x, \cdots q) \in \mathcal{S}$. Следовательно, $[r]_{i\cdots x\cdots k\cdots q} = 1$ и $[s]_{i\cdots j\cdots x\cdots q} = 1$. Из этого следует, что хотя бы для одного $x \in (1 : n)$ будет верно, что $[r]_{i\cdots x\cdots k\cdots q} \wedge [s]_{i\cdots j\cdots x\cdots q} = 1$, тогда $\bigvee_{x=1}^{n}[r]_{i\cdots x\cdots k\cdots q} \wedge [s]_{i\cdots j\cdots x\cdots q} = 1$. По определению $\vee$ получаем, что $[rs]_{i\cdots j\cdots k\cdots q} = 1$.

□

Рассмотрим пример. Матрица A размерности $4 \times 2 \times 3$ задает отношение «студенты-курсы-задачи»: первая размерность (4) представляет количество студентов, вторая размерность (2) – количество курсов, а третья размерность (3) – количество задач в каждом курсе. Элемент $A(i, j, k)$ содержит информацию о выполнении студентом с индексом i задачи с индексом k в курсе с индексом j. Например, если $A(0, 1, 0) = 1$, это может означать, что первый студент выполнил первую задачу во втором курсе.

$$\mathrm{A} = \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \right]$$

Аналогично, матрица B размерности $4 \times 3 \times 2$ задает отношение «студенты-проекты-задачи»: первая размерность (4) представляет количество студентов, вторая размерность (3) – количество проектов, а третья размерность (2) – количество задач в каждом проекте. Элемент $B(i, j, k)$ содержит информацию о доступности задачи с индексом k в проекте с индексом j для студента с индексом i. Например, если $B(2, 1, 0) = 1$, это может означать, что у третьего студента есть доступ к первой задаче

второго проекта.

$$\mathrm{B} = \left[\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \right]$$

Результат композиции отношений, то есть умножения матриц A и B, будет представлять отношение «студенты-курсы-проект» и будет представлено матрицей C размерности $4 \times 2 \times 2$. Элемент $C(i, j, k)$ будет содержать информацию, связанную с тем, как студент с индексом i применяет знания, полученные в курсе с индексом j, в проекте с индексом k.

$$\mathrm{C} = \left[\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right]$$

Таким образом, результат умножения многомерных матриц будет представлять собой композицию отношений, которые они задают. Это связано с тем, что умножение многомерных матриц является способом комбинирования отношений между элементами исходных матриц, чтобы получить новые связи между элементами в результирующей матрице.

3.1.2. Операции над многомерными матрицами

Создание многомерных матриц. Операция $generateMatrix$ создаёт новую многомерную матрицу с заданными размерностями и заполняет её элементы согласно заданной функции или правилу. Предположим, что у нас есть функция генерации $g(i_1, i_2, \cdots, i_k)$, которая возвращает значение для каждого элемента матрицы в зависимости от его индексов.

Формально, операция $generateMatrix(n_1, n_2, ..., n_k, g)$ определена следующим образом:

Создаётся многомерная матрица M размером $n_1 \times n_2 \times ... \times n_k$, где k – количество измерений, а n_i — размер i-го измерения. Для каждого элемента матрицы $M[i_1][i_2]...[i_k]$ с индексами $0 \leq i_j < n_j$ для всех $1 \leq j \leq k$, значение элемента задаётся как $M[i_1][i_2]...[i_k] = g(i_1, i_2, ..., i_k)$. Таким образом, каждый элемент $M[i_1][i_2]...[i_k]$ инициализируется значе-

нием, возвращаемым функцией g, которая может быть чистой функцией, зависящей только от индексов элемента, или может опираться на внешние данные или состояние для определения значения каждого элемента.

Примером может служить функция g, возвращающая 0 для всех элементов, тогда $generateMatrix$ создаст нулевую матрицу заданных размерностей. Если функция g возвращает $i_1 + i_2 + ... + i_k$, то матрица будет заполнена суммой индексов каждого элемента.

Сложение матриц представляет собой операцию поэлементного сложения соответствующих элементов входных матриц. Для каждой пары соответствующих элементов из двух матриц выполняется сложение, и результат становится соответствующим элементом результирующей матрицы.

Формально, если у нас есть две многомерные матрицы A и B одинаковых размеров, скажем, $n_1 \times n_2 \times \ldots \times n_k$, то каждый элемент $C_{d_1,d_2,...,d_k}$ результирующей матрицы C получается путем сложения соответствующих элементов $A_{d_1,d_2,...,d_k}$ и $B_{d_1,d_2,...,d_k}$:

$C_{d_1,d_2,...,d_k} = A_{d_1,d_2,...,d_k} + B_{d_1,d_2,...,d_k}$

Этот процесс повторяется для каждого элемента входных матриц, и таким образом выполняется сложение многомерных матриц произвольного размера.

Изменение структуры многомерной матрицы.

- **Операция среза для многомерных матриц** – это процесс извлечения подматрицы из исходной многомерной матрицы путём фиксации некоторых измерений на определенных индексах. Формально это можно описать следующим образом:

  Пусть дана многомерная матрица A размеров $n_1 \times n_2 \times \ldots \times n_k$, где k – количество измерений матрицы, а n_i – размерность по i-тому измерению. Срез матрицы A по i-тому измерению на j-м индексе, где $1 \leq j \leq n_i$, обозначается как $A_{i=j}$ и определяется следующим образом:

  $A_{i=j} = a_{d_1,d_2,...,d_{i-1},j,d_{i+1},...,d_k} \mid 1 \leq d_m \leq n_m, \forall m \neq i$

Здесь $a_{d_1,d_2,\ldots,d_k}$ — элементы исходной матрицы A, а $A_{i=j}$ – подматрица, полученная путём фиксации i-того измерения на j-м индексе и варьирования всех остальных индексов в их допустимых пределах.

Если необходимо выполнить срез по нескольким измерениям одновременно, операция среза применяется последовательно по каждому из указанных измерений.

- **Операция flatten** преобразует многомерную матрицу в одномерный вектор. Формально, если A – это многомерная матрица размеров $n_1 \times n_2 \times \ldots \times n_k$, операция $flatten(A)$ в порядке по строкам преобразует матрицу в одномерный вектор B следующим образом:

  Элементу $B[i]$, где $0 \leq i < n_1 \cdot n_2 \cdot \ldots \cdot n_k$, соответствует элемент матрицы $A[d_1][d_2]\ldots[d_k]$, где $i = d_1 \cdot (n_2 \cdot n_3 \cdot \ldots \cdot n_k) + d_2 \cdot (n_3 \cdot \ldots \cdot n_k) + \ldots + d_k$, d_j – индекс по j-му измерению, такой что $0 \leq d_j < n_j$ для всех $1 \leq j \leq k$.

  Индексы $d_1, d_2, \ldots, d_k$ получаются путём "разворачивания" индекса i в многомерное пространство индексов матрицы A. Сначала изменяется индекс d_k (последний элемент), и когда он достигает своего максимального значения n_k, он обнуляется, и происходит инкремент индекса d_{k-1}, и так далее вплоть до индекса d_1.

- **Операция reshape** преобразует одномерный вектор или многомерную матрицу в новую многомерную матрицу с заданными размерностями, при условии, что общее количество элементов остаётся неизменным.

  Формально, если B – это одномерный вектор длины m или многомерная матрица с общим количеством элементов m, то $reshape(B, n_1, n_2, \ldots, n_k)$ – это многомерная матрица C размеров $n_1 \times n_2 \times \ldots \times n_k$, где $n_1 \times n_2 \times \ldots \times n_k = m$. Элементы вектора B заполняют матрицу C последовательно согласно выбранному порядку индексации.

Операции set и get. Они используются для многомерных матриц для записи и извлечения значений элементов матрицы соответственно. Для формального описания предположим, что у нас есть многомерная матрица: A размером $n_1 \times n_2 \times \ldots \times n_k$, где k – количество измерений, а n_i – размер i-го измерения.

- **Операция** $get(A, i_1, i_2, \ldots, i_k)$**.** Для извлечения значения из матрицы по индексам $(i_1, i_2, \ldots, i_k)$ определена следующим образом:

  Проверяется, что каждый индекс i_j удовлетворяет условиям $0 \leq i_j < n_j$ для всех $1 \leq j \leq k$. Значение, которое извлекается, это элемент матрицы $A[i_1][i_2]\ldots[i_k]$.

- **Операция** $set(A, i_1, i_2, \ldots, i_k, value)$**.** Для записи значения в матрицу по индексам $(i_1, i_2, \ldots, i_k)$ с значением $value$ определена следующим образом:

  Проверяется, что каждый индекс i_j удовлетворяет условиям $0 \leq i_j < n_j$ для всех $1 \leq j \leq k$. Элемент матрицы по указанным индексам устанавливается равным $value$, то есть $A[i_1][i_2]\ldots[i_k] = value$.

Обе операции требуют знания структуры матрицы и правильности индексов, и они обрабатывают индексы в соответствии с размерностями матрицы. Неправильные индексы (например, если индекс выходит за допустимые границы) приводят к ошибке индексации.

3.2. Транслятор

Транслятор для языка Datalog является ключевой компонентой системы, предназначенной для обработки запросов с использованием многомерных матричных структур. Он выполняет преобразование логических выражений, написанных на Datalog, в операции над матрицами, что позволяет использовать алгоритмы линейной алгебры для эффективного выполнения запросов.

3.2.1. Процесс трансляции

Процесс трансляции Datalog в матричные операции представляет собой метод выполнения логических запросов, позволяющий тем самым воспользоваться преимуществами оптимизированных вычислительных библиотек и аппаратного ускорения, доступного для матричных операций. Процесс трансляции обеспечивает базу для создания высокопроизводительных систем обработки запросов, которые могут справляться с большими объемами данных и сложными запросами, характерными для современных приложений в области больших данных и искусственного интеллекта.

1. **Определение доменов для предикатов (Domain Map Construction):** Домены для всех предикатов определяются заранее на основе известных фактов и правил в программе. Это означает, что перед выполнением запросов домены каждого предиката строятся из набора уникальных значений, которые появляются в соответствующих позициях предикатов во всех фактах и возможных выводах правил.

2. **Инициализация матриц (Matrix Initialization):** Используя map доменов, для каждого предиката инициализируется матрица. Эти матрицы заполняются нулями и служат начальным представлением фактов и правил в матричной форме. Размер матрицы соответствует количеству термов, связанных с предикатом.

3. **Трансляция фактов и правил (Translation of Facts and Rules):** Каждый факт в Datalog преобразуется в одну матрицу, а правила разбираются на отдельные матрицы для каждого предиката в теле правила. Эти матрицы представляют собой входные данные для матричных операций.

4. **Матричное умножение (Matrix Multiplication):** После того как матрицы были инициализированы, они умножаются для вычисления результатов правил Datalog. Умножение матриц выпол-

няется в соответствии с порядком появления предикатов в теле правила.

5. **Согласование размерностей (Dimensionality Alignment):** Перед умножением каждой пары матриц необходимо удостовериться, что их размерности согласованы. Если размерности не соответствуют друг другу, вносятся корректировки, добавляя недостающие измерения в матрицы.

6. **Корректировка результатов (Result Adjustment):** Результат умножения может содержать избыточные измерения, которые не соответствуют ожидаемой структуре головного атома правила. В этом случае выполняется операция среза, чтобы убрать лишние измерения и адаптировать результат умножения к размерностям головного терма. В случаях, когда результат умножения матриц имеет меньшую размерность, чем ожидается в головном терме, необходимо увеличить размерность матрицы-результата. Это может быть достигнуто путем добавления измерений.

7. **Сложение матриц и дизъюнкция результатов (Matrix Addition and Disjunction of Results):** Как только все матрицы для головного атома правила (head atom) подготовлены, они складываются вместе, что соответствует дизъюнкции результатов в логическом выражении. Это означает, что если одно и то же утверждение может быть выведено несколькими способами (через разные правила), результаты каждого из этих способов комбинируются путем сложения соответствующих матриц. Результатом является консолидированная матрица для каждого уникального головного атома, которая включает все возможные выводы, полученные в ходе исполнения Datalog-программы.

3.2.2. Доказательство сохранения семантики

Изучение процесса преобразования конструкций Datalog в операции, применяемые к многомерным матрицам, требует тщательного анализа

для обеспечения сохранения семантики. В этом контексте выявляются ключевые параметры, которые станут основой для оценки точности трансляции, отражая соответствие между логическими операциями Datalog и их матричными аналогами.

Чтобы доказать, что каждое правило трансляции сохраняет семантику, можно использовать индукционный метод. Предположем, что было доказано, что композиция многоместных отношений эквивалентна умножению многомерных матриц, и теперь нам нужно показать, что для программы на Datalog и соответствующих матричных операций результат останется эквивалентным.

Сначала определяем, что значит сохранение семантики в контексте программ на Datalog и матричных операций: если начальное состояния базы данных DB_0 для Datalog равно начальному состоянию матрицы M_0 для матричной системы так, что каждое отношение в DB_0 находит своё представление в виде матрицы в M_0, то для каждого последующего состояния DB_i полученного путём применения оператора T_P в программе на Datalog, существует эквивалентное матричное состояние M_i, такое что DB_i соответствует M_i.

База индукции:

Инициализация матриц (Matrix Initialization): Инициализация матриц начинается с нулевого состояния M_0, что эквивалентно начальному состоянию базы данных DB_0. Поскольку каждый факт в Datalog является независимым и не влияет на другие факты, их соответствующие ячейки в матрице могут быть заполнены независимо, что позволяет нам точно отразить начальное состояние DB_0 в матричном представлении M_0, не теряя при этом семантики исходных данных.

Индукционное предположение:

Предположим, что после произвольного количества итераций k, DB_k соответствует M_k.

Индукционный переход:

- **Трансляция фактов и правил (Translation of Facts and Rules):** Каждый факт и каждое правило в DB_k эквивалентно

матрице в M_k. Правила трансляции работают одинаково для каждой итерации, следовательно, если факты эквивалентны в базовом случае, то правила трансляции будут также сохранять эквивалентность и для всех последующих DB_i и M_i.

- **Матричное умножение (Matrix Multiplication):** Результат применения правила r в Datalog, где все предикаты в теле правила удовлетворены, эквивалентен умножению соответствующих матриц в соответствии с порядком появления предикатов. По доказанному ранее получаем, что композиция отношений эквивалентна умножению матриц, результат применения T_P в каждом шаге будет иметь эквивалент в матричном произведении, что удовлетворяет индукционному предположению.

- **Согласование размерностей (Dimensionality Alignment) и Корректировка результатов (Result Adjustment):** Эти шаги обеспечивают, что результаты умножения матриц адаптируются к размерностям головных термов в Datalog и что лишние измерения убираются. Поскольку эти операции также могут быть выполнены на уровне отношений в Datalog, сохранение семантики гарантируется.

- **Сложение матриц и дизъюнкция результатов (Matrix Addition and Disjunction of Results):** Пусть $H_1, H_2, \ldots, H_n$ представляют матрицы, соответствующие головным атомам, выведенным по различным правилам для одинакового головного предиката h в состоянии базы данных DB_k.

  $M_{k+1}(h) = M_k(h) + H_1 + H_2 + \ldots + H_n$, где $M_{k+1}(h)$ и $M_k(h)$ – это состояния головной матрицы для предиката h после и до k-й итерации соответственно. Для каждого правила r_i, которое выводит головной атом h, соответствие H_i находится в M_k. Поскольку эти правила могут быть независимыми, их результаты, как условия дизъюнкции, могут быть объединены в один головной атом h. Таким образом, $M_{k+1}(h)$ представляет собой дизъюнкцию всех воз-

можных способов вывода атома h. Это позволяет сохранить эквивалентность результатов как в матричной, так и в логической форме.

Завершение индукции:

После всех итераций и применений правил Datalog, когда достигнута неподвижная точка F_P, это означает, что применение становится идемпотентным (то есть $T_P(DB_i) = DB_i$). На уровне матриц, это соответствует состоянию, когда дальнейшее умножение матриц не приводит к изменениям (то есть $M_{i+1} = M_i$). В таком случае, у нас есть матрица M_{lpf}, которая эквивалентна неподвижной точке DB_{lpf} в Datalog.

4. Реализация

4.1. Представление матриц и библиотека операций над многомерными матрицами

В основе библиотеки лежит тип данных Matrix, который является рекурсивной структурой, способной представлять матрицы произвольной размерности. Этот тип данных поддерживает как листовые элементы, так и вложенные матрицы. Он обеспечивает гибкость и масштабируемость представления, что критически важно для обработки сложных структур данных, характерных для Datalog.

Библиотека предоставляет ряд функций для создания, трансформации и манипуляции матрицами. Ключевые операции включают генерацию матриц, изменение их формы, доступ и модификацию элементов, а также получение подматриц. Эти операции являются основой для реализации более сложных алгоритмов, таких как матричное умножение, которое используется для композиции отношений в Datalog.

Библиотека операций над многомерными матрицами тесно интегрирована с транслятором Datalog, обеспечивая необходимую функциональность для преобразования логических структур в матричные. Она играет важную роль в оптимизации запросов и эффективности выполнения программ, написанных на Datalog.

После разработки библиотеки операций над многомерными матрицами было проведено тестирование кода с использованием библиотеки HUnit [1]. Были написаны юнит-тесты для проверки корректности всех ключевых функций. Тестирование позволило убедиться в надежности и стабильности библиотеки, гарантируя правильное функционирование интеграции с транслятором Datalog и обеспечивая точность выполнения запросов на языке Datalog.

[1] https://hackage.haskell.org/package/HUnit (дата обращения: 2023-12-21)

4.2. Транслятор

Была реализована первоначальная концепция транслятора Datalog, который конвертирует логические запросы в матричные операции для последующего улучшения производительности выполнения.

Во время реализации транслятора возникла проблема, обусловленная неочевидностью композиции отношений разной арности. Проблема заключалась в том, что матрицы, представляющие отношения с различной арностью, имеют разную размерность, что затрудняло выполнение матричных операций. Решением стало увеличение размерности матриц, соответствующих отношениям с меньшей арностью, до размерности матриц, представляющих отношения с большей арностью. Это позволило реализовать корректное умножение многомерных матриц, при котором сохраняется семантика исходных отношений.

4.3. Интерпретатор Datalog для проверки эквивалентности

Был разработан интерпретатор, который в соответствии с заданной семантикой неподвижной точки Datalog производит интерпретацию этого языка. Для тестирования была использована библиотека Hengehog[2], которая помогает рандомно генерировать AST Datalog. После этого происходит интерпретация полученного AST и трансляция в операции над многомерными матрицами. В конечном итоге результаты сравниваются для проверки правильности интерпретации и трансляции.

[2] https://hedgehog.qa/ (дата обращения: 2023-12-20)

Листинг 1: Псевдокод процесса трасляции Datalog в операции над многомерными матрицами.

```
translate datalogProgram =
  let
    domainMap = createDomainMap datalogProgram
    matrices = initializeMatrices domainMap
    updatedMatrices = foldl (\mats fact ->
      updateMatrix mats fact.predicate (translateFact fact)
    ) matrices datalogProgram
    intermediateMatrices = foldl (\mats rule ->
      let bodyMatrices = map (\p -> mats[p]) rule.body
          result = head bodyMatrices
          multipliedResult = foldl (\res idx ->
            let updatedRes = matrixMultiplication res
            (bodyMatrices[idx])
            in alignDimensions updatedRes
          ) result [1..length bodyMatrices]
          adjustedResult = adjustResult multipliedResult
          disjunctiveResult = matrixAddition
          mats[rule.head.predicate] adjustedResult
      in updateMatrix mats rule.head.predicate
      disjunctiveResult
    ) updatedMatrices datalogProgram
  in intermediateMatrices
```

Заключение

В рамках работы были выполнены следующие задачи.

- Была разработана библиотека для операций с многомерными матрицами, которая обеспечивает необходимые функции для хранения матриц и выполнения над ними операций. Эта библиотека была тщательно протестирована, чтобы гарантировать её надёжность для последующего использования в процессе трансляции;

  Ссылка на GitHub репозиторий — https://github.com/ALanovaya/datalog-translator

- Реализованный транслятор успешно выполняет преобразование Datalog-запросов в операции над многомерными матрицами. Он точно следует семантике языка Datalog, что было подтверждено сравнением результатов его работы с результатами интерпретатора Datalog;

- Интерпретатор для Datalog был написан с целью проверки результатов, получаемых транслятором. Сравнение результатов, полученных интерпретатором и транслятором, показало их полное соответствие, подтверждая тем самым корректность трансляции;

- Тестирование всех разработанных компонентов показало, что они работают стабильно и предоставляют точные результаты. Тесты подтвердили, что библиотека, транслятор и интерпретатор действуют согласованно и могут быть использованы для дальнейшей работы с Datalog и многомерными матрицами;

В следующем семестре планируется решить несколько задач.

- Оптимизация алгоритма трансляции;

- Оптимизация хранения матриц;

- Оптимизация вычисления операций над многомерными матрицами;

- Проведение анализа производительности;

Список литературы

[1] Arntzenius Michael, Krishnaswami Neelakantan R. Datafun: A Functional Datalog. — New York, NY, USA : Association for Computing Machinery, 2016. — sep. — Vol. 51. — P. 214–227. — URL: https://doi.org/10.1145/3022670.2951948.

[2] Bégay Pierre-Léo, Crégut Pierre, Monin Jean-François. Developing and Certifying Datalog Optimizations in Coq/Mathcomp // Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs. — New York, NY, USA : Association for Computing Machinery, 2021. — CPP 2021. — P. 163–177. — URL: https://doi.org/10.1145/3437992.3439913.

[3] Ceri S., Gottlob G., Tanca L. What you always wanted to know about Datalog (and never dared to ask). — 1989. — Vol. 1. — P. 146–166.

[4] Chaudhuri Surajit. An Overview of Query Optimization in Relational Systems // Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. — New York, NY, USA : Association for Computing Machinery, 1998. — PODS '98. — P. 34–43. — URL: https://doi.org/10.1145/275487.275492.

[5] A Comparison of Approaches to Large-Scale Data Analysis / Andrew Pavlo, Erik Paulson, Alexander Rasin et al. // Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. — New York, NY, USA : Association for Computing Machinery, 2009. — SIGMOD '09. — P. 165–178. — URL: https://doi.org/10.1145/1559845.1559865.

[6] Crowdsourcing Cybersecurity: Cyber Attack Detection Using Social Media / Rupinder Paul Khandpur, Taoran Ji, Steve Jan et al. // Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. — New York, NY, USA : Association for Computing Machinery, 2017. — CIKM '17. — P. 1049–1057. — URL: https://doi.org/10.1145/3132847.3132866.

[7] Custodiol for myocardial protection and preservation: a systematic review / J. James B. Edelman, Michael Seco, Ben Dunne et al. — 2013. — Vol. 2. — URL: https://www.annalscts.com/article/view/2881.

[8] Datalog and Recursive Query Processing / Todd J. Green, Shan Shan Huang, Boon Thau Loo, Wenchao Zhou. — 2013. — Vol. 5. — P. 105–195. — URL: http://dx.doi.org/10.1561/1900000017.

[9] Design and Implementation of the LogicBlox System / Molham Aref, Balder ten Cate, Todd J. Green et al. // Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. — New York, NY, USA : Association for Computing Machinery, 2015. — SIGMOD '15. — P. 1371–1382. — URL: https://doi.org/10.1145/2723372.2742796.

[10] Feichtinger Oskar, McAllister B. L. Binary Relations as Boolean Matrices. — Taylor Francis, 1970. — Vol. 43. — P. 8–14. — https://doi.org/10.1080/0025570X.1970.11975988.

[11] Jordan Herbert, Scholz Bernhard, Subotić Pavle. Soufflé: On Synthesis of Program Analyzers // Computer Aided Verification / Ed. by Swarat Chaudhuri, Azadeh Farzan. — Cham : Springer International Publishing, 2016. — P. 422–430.

[12] Liu Bozhen, Huang Jeff. D4: Fast Concurrency Debugging with Parallel Differential Analysis. — New York, NY, USA : Association for Computing Machinery, 2018. — jun. — Vol. 53. — P. 359–373. — URL: https://doi.org/10.1145/3296979.3192390.

[13] Neo: a learned query optimizer / Ryan Marcus, Parimarjan Negi, Hongzi Mao et al. — Association for Computing Machinery (ACM), 2019. — . — Vol. 12. — P. 1705–1718. — URL: http://dx.doi.org/10.14778/3342263.3342644.

[14] Ross Kenneth A. Modular Stratification and Magic Sets for DATALOG Programs with Negation. // PODS / Ed. by Daniel J. Rosenkrantz,

Yehoshua Sagiv. — ACM Press, 1990. — P. 161–171. — URL: http://dblp.uni-trier.de/db/conf/pods/pods90.html#Ross90.

[15] Sato Taisuke. A Linear Algebraic Approach to Datalog Evaluation. — 2016. — Vol. abs/1608.00139. — arXiv : 1608.00139.

[16] Solo Ashu. Multidimensional Matrix Mathematics: Multidimensional Matrix Equality, Addition, Subtraction, and Multiplication, Part 2 of 6. — 2010. — 06. — Vol. 2185.

[17] Using Datalog with Binary Decision Diagrams for Program Analysis / John Whaley, Dzintars Avots, Michael Carbin, Monica S. Lam // Programming Languages and Systems / Ed. by Kwangkeun Yi. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2005. — P. 97–118.