Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

ДИССЕРТАЦИЯ
на соискание ученой степени
МАГИСТРА

Тема: **Исследование и разработка методов выявления синтаксической близости математических выражений в формате MathML**

Студент гр. 63501/2 М.А. Пономарев

Санкт-Петербург – 2016

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Диссертация допущена к защите
зав. кафедрой

_____ В.М. Ицыксон

«____» _____ 2016 г.

ДИССЕРТАЦИЯ
на соискание ученой степени МАГИСТРА

Тема: **Исследование и разработка методов выявления синтаксической близости математических выражений в формате MathML**

Направление: 09.04.01 – Информатика и вычислительная техника
Магистерская программа: 09.04.01.14 – Проектирование аппаратно–программных средств вычислительной техники

Выполнил студент гр. 63501/2 _____ М.А. Пономарев

Научный руководитель,
к. т. н., доц. _____ Е.В. Пышкин

Консультант по нормоконтролю,
ст. преподаватель _____ С.А. Нестеров

Санкт-Петербург – 2016

УТВЕРЖДАЮ
зав. кафедрой
_____ В.М. Ицыксон
«____»_____ 2016 г.

ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ
студенту Пономареву Михаилу Александровичу

1. Тема проекта (работы)
   «Исследование и разработка методов выявления синтаксической близости математических выражений в формате MathML»

2. Срок сдачи студентом законченного проекта (работы)
   3 июня 2016г.

3. Исходные данные к проекту (работе)
   Нет

4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

   - Введение
   - Исследование предметной области
   - Разработка алгоритма для поиска синтаксической схожести математических выражений в формате MathML
   - Разработка программного обеспечения на основе разработанного алгоритма для поиска синтаксически схожих математических выражений
   - Оценка эффективности разработанного алгоритма
   - Заключение

5. Перечень графического материала (с точным указанием обязательных чертежей)
   Нет

6. Консультанты по проекту (с указанием относящихся к ним разделов проекта работы)
   Нестеров С.А. — консультант по нормоконтролю

7. Дата выдачи задания
   1 сентября 2015г.

Руководитель к.т.н., доц. Е.В. Пышкин _____
Задание принял к исполнению, М.А. Пономарев _____

РЕФЕРАТ

Отчет, 80 стр., 32 рис., 30 табл., 15 ист., 1 прил.

MATHML, СХОЖЕСТЬ МАТЕМАТИЧЕСКИХ ВЫРАЖЕНИЙ, СИНТАКСИЧЕСКАЯ СХОЖЕСТЬ, СТРУКТУРНАЯ СХОЖЕСТЬ, СХОЖЕСТЬ ПОДВЫРАЖЕНИЙ

Целью данной работы является исследование существующих решений и разработка новых алгоритмов по поиску синтаксической схожести между математическими выражениями, представленными в формате MathML. В настоящее время существует лишь один алгоритм, получивший широкое применение в области оценки систем по распознаванию рукописной математической нотации, который предоставляет возможность определить дистанцию редактирования, чтобы превратить одно выражение в другое, и не предоставляет никакой дополнительной информации о схожести двух выражений.

В первой главе содержатся общие сведенья о способах определения схожести между упорядоченными ориентированными графами.

Во второй главе дается определение паттернам синтаксической схожести выражений в формате MathML и приводится модификация одного из исследованных алгоритмов для определения схожести между выражениями.

В третьей главе описан процесс разработки приложения, реализующего поиск схожести математических выражений на основе разработанного алгоритма.

В четвертой главе приведены общие сведения о полученных результатах, приведена оценка разработанного алгоритма и рекомендации к его применению.

ABSTRACT

Report, 80 pages, 32 figures, 30 tables, 15 references, 1 appendicies

MATHML, SYNTACTIC SIMILARITY, STRUCTURAL SIMILARITY, SUBEXPRESSION SIMILARITY,

The aim of this work is to study existing solutions and to develop new algorithms for searching the syntactic commonality between the mathematical expressions presented in the MathML format. Nowadays there is only one widely used algorithm in the evaluation systems of handwritten mathematical notation recognition, which provides the possibility to define the edit distance to transform one expression into another and doesn‘t contribute any additional information about the commonality of two expressions.

The first chapter contains general information about the ways of the commonality determination between the ordered directed graphs.

The second chapter defines the syntactical commonality patterns of the expressions in MathML format and provides the modification of the investigated algorithm for determining commonality between the expressions.

The third chapter describes the process of an application development that implements the search of the mathematical expressions commonality on the basis of the developed algorithm.

The fourth chapter presents the results overview, the evaluation of developed algorithm and recommendations for use.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В наши дни поисковые системы в интернете являются неотъемлемыми и необходимыми инструментами. К сожалению, поиск математической нотации до сих пор сопряжен с рядом трудностей. Они состоят и в нетривиальной математической разметке математических выражений и формул, затрудняющей формулировку запроса пользователем, и в отсутствии унифицированной цифровой математической библиотеки. До сих пор нет общепринятого соглашения о том, какой формат хранения математической нотации использовать. Из–за этого многие поисковые машины сталкиваются с проблемой конвертации различных форматов в один.

Один из наиболее распространенных языков для представления математических выражений в интернете является MathML, рекомендованный математической группой W3C. Многие поисковые системы используют этот язык для хранения математической нотации. В основе практически всех таких систем лежит поиск по точному совпадению между выражениями или их частями, и не предусматривается или слабо развит поиск схожих математических выражений. Из всех рассмотренных алгоритмов поиска схожих математических выражений [1], [2], каждый имеет ряд существенных недостатков в виде плохой точности поиска по большой базе выражений или в виде нацеленности на решение узкого круга задач.

Синтаксическая схожесть математических выражений до сих пор определена нечетко — нет общепринятых паттернов, которые определяли бы схожесть двух выражений, полагаясь на те или иные критерии. В то же время разработка поисковой системы, которая реализовала бы возможность подобного рода поиска, была бы очень актуальна при сортировке однотипных заданий в базе заданий единых государственных экзаменов ОГЭ и ЕГЭ или поиске похожих заданий из сторонних учебных ресурсов с целью того, чтобы учащийся или абитуриент мог систематичнее и тщательнее подготовиться к заданиям определенного вида. Кроме этого, система могла бы найти полезное применение при формировании электронных методических пособий или поиску выражений по шаблону, когда пользователь не знает точного выражения, но помнит его форму, синтаксическую структуру.

Целью данной работы является разработка алгоритма определения синтаксической схожести выражений в формате MathML и его

реализация в виде программного комплекса для последующей оценки его эффективности.

СЛОВАРЬ ТЕРМИНОВ

XML (от англ. eXtensible Markup Language) — расширяемый язык разметки документов, используемых в сети интернет.

MathML (от англ. Mathematical Markup Language) — язык разметки на основе XML для представления математических символов и формул в документах сети интернет.

EMERS (от англ. a tree matching-based performance evaluation metric for mathematical expression recognition) — метрика определения схожести распознанного математического выражения с действительным.

Нотация — система условных обозначений, принятая в какой-либо области знаний или деятельности. Включает множество символов, используемых для представления понятий и их взаимоотношений, составляющее алфавит нотации, а также правила их применения.

Нормализация математического выражения — процесс удаления или замены некоторой неважной информации в древовидном представлении математического выражения, такой как, к примеру, название переменных или численных значений.

Терминальный узел — узел, не имеющий дочерних элементов.

1. ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание структуры математических выражений в формате MathML

MathML (Mathematical Markup Language)— это язык разметки на основе XML для представления математических символов и формул. Логическая структура документа схожа с логической структурой документа XML: он имеет обязательный корневой элемент, включающий в себя вложенные в него один или несколько элементов. Граница элемента определяется знаками < и >. Элемент может содержать тег, определяющий имя элемента, а также один или несколько атрибутов и данные элемента.

Существует два вида разметки MathML —- Presentation Markup и Content Markup. Каждый имеет свой набор возможных элементов и разный принцип их вложения друг в друга. Для того, чтобы лучше понять, какой из видов разметки использовать, рассмотрим их подробнее в следующих разделах на основе документации [3] по MathML.

1.1.1. Разметка Presentation Markup

Разметка Presentation Markup предназначена, чтобы описать визуальное представление выражения без учета его семантики. Этот вид разметки имеет небольшое фиксированное количество тегов, каждый из которых, в большинстве случаев, отвечает за конкретный структурный элемент в выражении, к примеру, операции деления или арифметического квадратного корня. Каждый элемент имеет фиксированное или нет количество обязательных элементов, которые он должен содержать, к примеру, элемент, отвечающий за представление арифметического квадратного корня, должен содержать в себе элемент, отвечающий за выражение, которое будет находится под корнем. Элементы условно можно разделить на 4 группы, отвечающих за: лексемы (табл. 1.1), общую разметку выражений, визуально влияющих на синтаксическую структуру выражений (табл. 1.2) и нет (табл. 1.3), представление выражений с индексами (табл. 1.4), представление выражений в виде таблиц и матриц (табл. 1.5).

Таблица 1.1. Элементы для отображения лексем

Элемент	Описание
mi	Переменная
mn	Число
mo	Оператор, специальный символ, разделитель
mtext	Текст
mspace	Пробел
mglyph	Отображение символов без Unicode

Таблица 1.2. Элементы для общей разметки выражений, визуально влияющие на синтаксическую структуру выражения

Элемент	Описание	Кол–во аргумен-тов
mrow	Группир. подвыражений между собой	0+
mfrac	Деление одного выражение на другое	2
msqrt	Извлечение из выражения квадратичного корня	1
mroot	Извлечение из выражения корня из n-ой степени	2
mfenced	При перечислении множества или массива, возможность добавления пользовательских открывающих/закрывающих скобок и разделителя	0+

Основное назначение аттрибутов в Presentation Markup — задания параметров отображения элементов в выражении, таких как цвет, размер и задний фон. В силу того, что они не влияют на синтаксическую структуру выражения, рассматриваться подробнее они не будут.

Отдельно стоит отметить, что для Presentation Markup не предусмотрены отдельные элементы, отвечающие за отображение функций

Таблица 1.3. Элементы для общей разметки выражений, визуально не влияющие на синтаксическую структуру выражения

Элемент	Описание	Кол–во аргументов
mstyle	Отвечает за стиль отображаемого выражения	1*
merror	Вывод сообщений об ошибке в мат. выражении	1*
mpadded	Отступы между элементами в выражении	1*
mphantom	Скрытие элементы при отображении, но занимающие место	1*
menclose	Помещает выражение в рамки	1*
maction	Реакция на нажатие отображаемого элемента	1+

Таблица 1.4. Элементы для отображения выражений с индексами

Элемент	Описание	Кол–во аргументов
msub	Выражение с подстрочным индексом	2
msup	Выражение с надстрочным индексом	2
msubsup	Выражение с подстрочным и надстрочным индексом	3
munder	Нижняя фигурная скобка под выражением	2
mover	Верхняя фигурная скобка над выражением	2
munderover	Интегральная кривая с нижним и верхним пределом	3
mmultiscripts	Позволяет создавать 4 индекса в виде матрицы вокруг выражения	1+

Таблица 1.5. Элементы для представления выражений в виде таблиц и матриц

Элемент	Описание	Кол–во аргумен-тов
mtable	Таблица из n-го количества рядов	0+
mlabeledtr	Ряд с заглавным элементом	1+
mtr	Ряд в матрице	0+
mtd	Элемент в ряду	1*

как, к примеру, sin, cos, tg, ctg, ln и т.п.

1.1.2. Разметка Content Markup

Разметка Presentation Markup описывает математическое выражение без учёта его семантики. Разметка Content Markup разработана для того, чтобы помимо описания структуры выражения, иметь возможность описать его семантику. Во многих математических нотациях отображение одной и той же формулы может трактоваться по разному, и без дополнительной информации невозможно принять решение, как именно трактовать конкретное отображение примера. Трудности возникают из–за того, что разным представлениям математической нотации может соответствовать одна и та же семантика и наоборот. К примеру, конструкция «M перемножается с g» может быть в одном контексте представлена как «$M \times g$», а в другом «Mg». В последнем случае, отрывая от контекста этот пример, будет невозможно принять решение, является ли это, к примеру, именем химического элемента или изначально задуманной конструкцией.

Чтобы решить эту проблему, с помощью разметки Content Markup устанавливается связь между математической структурой выражения или переменной и их семантикой. Каждой математической структуре соответствует базовая семантика, которая может быть изменена. К примеру, чтобы изменить семантику переменной v и рассматривать ее как векторную величину, необходимо добавить аттрибут «type=vector» к элементу, отвечающему за отображение этой самой переменной.

Как и в разметке представления, в Content Markup в основе ле-

жит дерево выражения, точно также терминальные узлы представляются как базовые математические объекты, такие как числа, переменные или арифметические операции. Внутренние же элементы представляют собой более сложный объект, чем в Presentation Markup. В Content Markup гораздо больше элементов, которые условно можно разделить на следующие группы:

- **константы и символы**: cn, ci, csymbol;
- **конструкции выражений** — определяют сущность выражения или его части, будет ли оно, скажем, интервальным значением или выражением–условием: apply, interval, inverse, sep, condition и др.;
- **арифметика, алгебра, логические операции**: factorial, divide, max, min, minus, plus, power, rem, times, root, and, or, xor, not и др.;
- **отношения между выражениями**: eq, neq, geq, leg, euivalint, approx, factorof;
- **интегральные и другие вычисления**: int, diff, partialdiff, lowlimit, uplimit, bvar, degree, devergence, grad;
- **наборы**: set, list, union, intersect, in, notin, subset, prsubset и др.;
- **последовательности**: sequences and series: sum, product, limit;
- **классические функции**: exp, ln, log, sin, cos, tan;
- **элементы статистики**: mean, sdev, variance, median, mode, moment;
- **линейная алгебра**: vector, matrix, matrixrow, determinant, transpose, selector, vectorproduct;
- **семантическая взаимосвязь между элементами**: annotation, semantics, annotation-xml;
- **константы и символьные переменные**: constant and symbol elements: integers, reals, rationals, naturalnumbers, true, false, infinity и др.;

По сравнению с Presentation Markup, в случае с Content Markup будет сложнее предсказать, как в конечном итоге будет выглядеть математическое выражение. Кроме этого, некоторые операции, несущие разный семантический смысл при отображении могут выводиться одинаково или очень похоже.

1.2. Методы определения схожести упорядоченных деревьев

Математическое выражение, описываемое с помощью MathML, представляется в виде ориентированного упорядоченного дерева с неограниченным количеством дочерних узлов. Для того, чтобы в дальнейшем дать определение схожести двум математическим выражениям в формате MathML, в данном разделе рассматриваются методы определения схожести ориентированных упорядоченных деревьев.

Для большей наглядности алгоритмов определения схожести тем или иным методом все они будут рассматриваться на примере деревьев T_0, T_1 и T_2 (рис.1.1), для которых будет подсчитываться схожесть между деревом T_0 и деревьями T_1 и T_2.

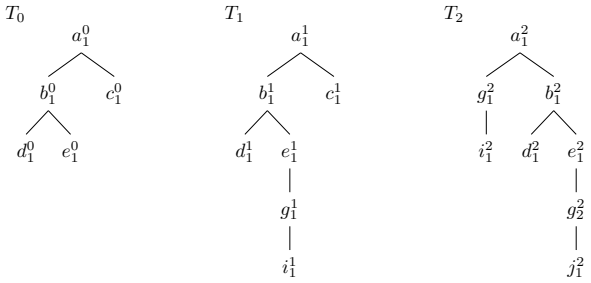

Рисунок 1.1. Упорядоченные ориентированные деревья T_0, T_1 и T_2

Верхние индексы у узлов деревьев на рис. 1.1 определяют дерево, в котором они находятся — так, к примеру, у дерева T_0 все верхние индексы имеют значение равным 0. Нижние индексы определяют, какими по счету встречаются одинаковые узлы в своем дереве — к примеру, у дерева T_2 есть два одинаковых узла g, первый из которых будет иметь нижний индекс равным 1, а второй равным 2. При исследовании некоторых методов индексы будут опускаться для большей

наглядности.

1.2.1. Метод, основанный на введении дистанции редактирования между деревьями

В работе [4] схожесть двух деревьев определяется дистанцией между двумя упорядоченными деревьями (Edit Tree Distance). Под дистанцией понимается минимальное количество операций редактирования, таких как вставка, удаление и изменение узлов, чтобы преобразовать одно дерево в другое.

Пусть S — последовательность $\{s_1, s_2, \ldots, s_k\}$ операций редактирования для преобразования одного дерева в другое.

Пусть γ — метрика дистанции между двумя деревьями, характеризующаяся неотрицательным весом для операции редактирования узла $a \to b$. Веса для разных узлов могут быть различными в зависимости от их значимости или расположения относительно корня дерева. На метрику вводятся следующие ограничения:

- $\gamma(a \to b) \geqslant 0$
- $\gamma(a \to b) = \gamma(b \to a)$

Применяя метрику дистанции к последовательности S получаем следующую формулу: $\gamma(S) = \sum_{i=1}^{|S|} \gamma(s_i)$. Тогда дистанция между двумя выражениями:

$$\delta(T_1, T_2) = min\{\gamma(S)\} \tag{1.1}$$

Применимо к рассматриваемым деревьям T_0, T_1 и T_2, преобразуем запрашиваемое дерево T_0 к дереву T_1 (рис. 1.2) и к дереву T_2 (рис. 1.3).

Как видно из рис. 1.2 и рис. 1.3, для того, чтобы преобразовать дерево T_0 к дереву T_1, требуется последовательность из двух операций добавления, а для того чтобы преобразовать дерево T_0 к дереву T_2 — последовательность из четырех операций: одной операции удаления, и трех операций добавления узлов.

Применяя формулу 1.1 с весами всех узлов равными 1, получаем схожесть между запрашиваемым деревом T_0 и деревьями T_1 и T_2, равными $S_{TED}(T_0, T_1) = 2$ и $S_{TED}(T_0, T_2) = 4$ соответственно.

Различные алгоритмы, в основе которых лежит данный метод, по-разному находят последовательность S_i, чтобы преобразовать одно

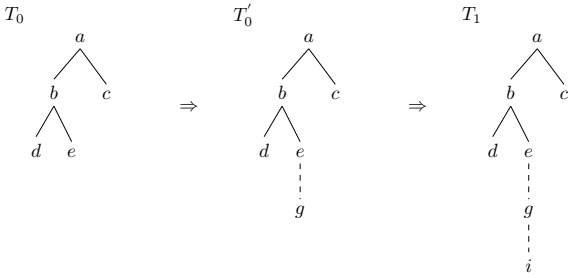

Рисунок 1.2. Приведение дерева T_0 к дереву T_1 с помощью операций редактирования

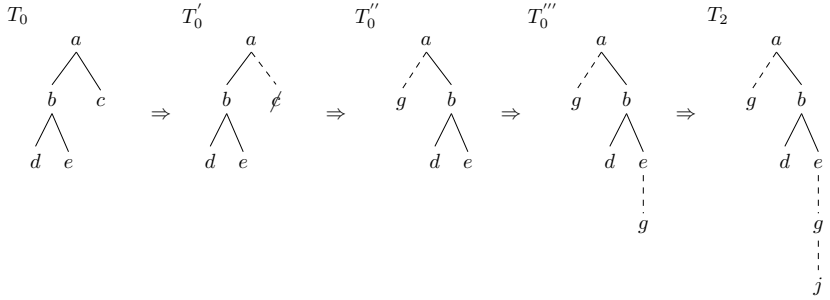

Рисунок 1.3. Приведение дерева T_0 к дереву T_2 с помощью операций редактирования

дерево к другому. В работах [4], [5], [6], [7] приведены четыре разных алгоритма по нахождению дистанции между двумя упорядоченными графами. При реализации последнего (RTED) учтены все слабые стороны ранних алгоритмов, и его можно считать, на момент написания диссертации, самым оптимальным. Алгоритмические сложности данных алгоритмов представлены в табл. 1.6.

1.2.2. Метод, основанный на наложении одного дерева на другое

Схожесть двух деревьев можно определить количеством общих узлов при наложении одного дерева на другое. В работе [8] приводится алгоритм на основе данного метода. Схожесть по данному алгорит-

Таблица 1.6. Сравнение алгоритмов, реализующих метод tree edit distance

Название алгоритма	Средняя скорость выполнения	Объем занимаемой памяти	Особенности
TED [4]	$O(n^4)$	$O(n^2)$	Эффективен для сбалансированных деревьев
ODTED [6]	$O(n^3)$	$O(n^2)$	Часто скорость выполнения ухудшается на сбалансированных деревьях
RTED [7]	$O(n^3)$	$O(n^2)$	Хорошо себя проявляет в любых ситуациях

му определяется количеством общих узлов с одинаковыми правилами перехода к последующим узлам при таком наложении одного дерева на другое, когда это количество будет наибольшим (Tree Overlapping). Правилом перехода является правило, по которому один узел переходит в какой–то другой или несколько других. Под наложением одного дерева на другое данным алгоритмом подразумевается наложение узла первого дерева на такой же узел во втором дереве. Последующие узлы деревьев при совпадении также накладываются друг на друга. Пример наложения T_0 на T_1 и T_0 на T_2 из рассматриваемых деревьев в схожих узлах продемонстрирован на рис. 1.4 и на рис. 1.5 соответственно.

В рассматриваемых примерах сразу были наложены «самые удачные» и единственные узлы, которые приводят к наибольшему количеству общих правил перехода между деревьями. В целом, таких вариантов наложений может быть много, в зависимости от того, какие узлы выбираются. В случае с наложением узла a_1^0 дерева T_0 на узел a_1^1 дерева T_1 — $C_{TO}(a_1^0, a_1^1) = 2$, а при наложении узла b_1^0 дерева T_0 на узел b_1^2 дерева T_2 схожесть будет равна $C_{TO}(b_1^0, b_1^2) = 1$.

Схожесть деревьев по этой метрике определяется следующей формулой:

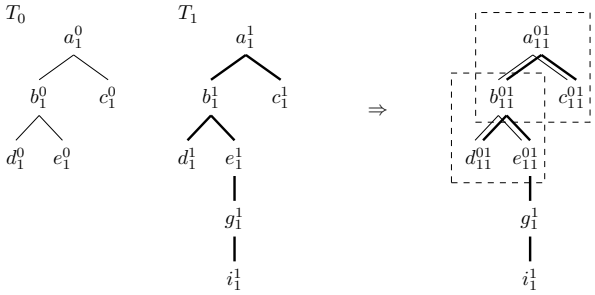

Рисунок 1.4. Пример наложения дерева T_0 и T_1 в узлах a_1^0 и a_1^1

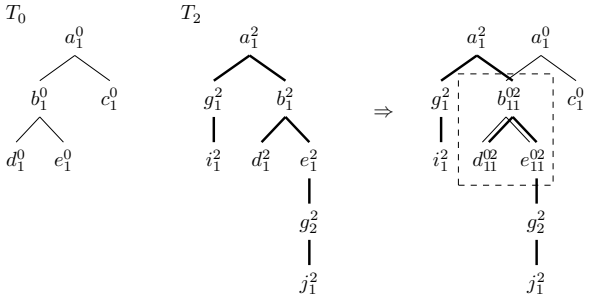

Рисунок 1.5. Пример наложения дерева T_0 и T_2 в узлах b_1^0 и b_1^1

$$S_{TO}(T_1, T_2) = \max_{n_1 \in NT(T_1), n_2 \in NT(T_2)} C_{TO}(n_1, n_2) \tag{1.2}$$

В формуле 1.2 $NT(T)$ — набор нетерминальных узлов в дереве T.

Схожесть между деревом T_0 и T_1 будет равна $S_{TO}(T_0, T_1) = 2$, а между деревом T_0 и T_2 равна $S_{TO}(T_0, T_2) = 1$.

При наложении узлов друг на друга между деревьями при поиске схожести этим методом они должны совпадать в названии и в правилах перехода к последующим узлам. Алгоритм по нахождению максимально схожего дерева дереву T_0 из корпуса деревьев можно ускорить, не вычисляя поочередно схожесть каждого дерева из набора с запрашиваемым деревом. Для этого необходимо составить таблицу индексов $I[p]$ для корпуса деревьев, по которым будем искать схожесть с запрашиваемым деревом. Приведем пример и составим такую

таблицу для корпуса деревьев T_1 и T_2 (рис. 1.1) — деревьях, рассматриваемых в качестве примера в этом разделе:

Таблица 1.7. Таблица индексов для корпуса деревьев T_1 и T_2

p	$I[p]$
$a \to b\,c$	$\{a_1^1\}$
$b \to d\,e$	$\{b_1^1, b_1^2\}$
$e \to g$	$\{e_1^1\}$
$g \to i$	$\{g_1^1, g_1^2\}$
$a \to g\,b$	$\{a_1^2\}$
$g \to j$	$\{g_2^2\}$

В левом столбце табл. 1.7 содержатся правила перехода от одного узла к другому(им). В правом — список узлов, содержащихся в деревьях корпуса с представленным в левом столбце правилом перехода. Из табл. 1.7 видно, что правилу перехода $b \to d\,e$ соответствуют сразу два узла из двух разных деревьев. Помимо этого, при рассмотрении других случаев, может быть такая ситуация, когда одному и тому же правилу перехода могут соответствовать сразу несколько узлов из одного и того же дерева. Составим такую же таблицу для запрашиваемого дерева T_0:

Таблица 1.8. Таблица индексов для дерева T_0

p	$I[p]$
$a \to b\,c$	$\{a_1^0\}$
$b \to d\,e$	$\{b_1^0\}$

После составления таблиц индексов $I[p]$, необходимо составить таблицу Top — Левый столбец в таблице будет указывать на совпавшие узлы между запрашиваемым деревом и деревьями корпуса, а правый — на совпавшие вышележащие узлы относительно узлов в левом столбике, при проходе до которых все промежуточные узлы в деревьях с индексом этих узлов тоже совпадают. Пример данной таблицы для рассматриваемых в качестве примера деревьев (рис. 1.1):

Таблица 1.9. Таблица Top между узлами дерева T_0 и узлами деревьев корпуса T_1 и T_2

(n, m)	$top(n, m)$
$\{a_1^0, a_1^1\}$	$\{a_1^0, a_1^1\}$
$\{b_1^0, b_1^1\}$	$\{a_1^0, a_1^1\}$
$\{b_1^0, b_1^2\}$	$\{a_1^0, a_1^2\}$

Далее, необходимо подсчитать $C[n, m]$, который будет показывать количество наложенных друг на друга узлов начиная с узлов n и m с использованием таблиц индексов (табл. 1.7 и табл. 1.8) и таблицы top (табл. 1.9) для деревьев корпуса.

Для этого применяется следующий алгоритм:

1. Находим по таблице индексов из столбца $I[p]$ одинаковые узлы с узлами запрашиваемого дерева T_0, положим пара таких узлов будет (n, m);

2. ищем по таблице *top* вышележащие узлы, по пути к которым все промежуточные узлы совпадают, положим эти узлы (n', m');

3. увеличиваем значение $C(n', m')$ на 1.

После применения данного алгоритма получаем следующие результаты: $C[a_1^0, a_1^1] = 2$, $C[a_1^0, a_1^2] = 1$

Преобразуем формулу 1.2 для ее применения для оценки схожести между запрашиваемым деревом T_0 и деревом корпуса:

$$S_{TO}(T_0, T) = \max_{n \in NT(T_0), m \in NT(T)} C[top(n, m)] \tag{1.3}$$

Применяя эту формулу 1.3 получаем схожесть между запрашиваемым деревом T_0 и деревьями T_1 и T_2 равными $S_{TO}(T_0, T_1) = 2$ и $S_{TO}(T_0, T_2) = 1$ соответственно.

При таком подходе объем занимаемой памяти при предварительной индексации корпусов деревьев составляет алгоритмическую сложность порядка $O(PR)$, где PR — количество правил перехода в таблице индексов.

1.2.3. Метод, основанный на подсчете количества общих подпутей между деревьями

В работе [8] схожесть двух деревьев определяется количеством общих подпутей между ними (Subpath Set). Подпутем является путь от корневого узла или одного из узлов в дереве до терминального или до другого узла по пути к нему.

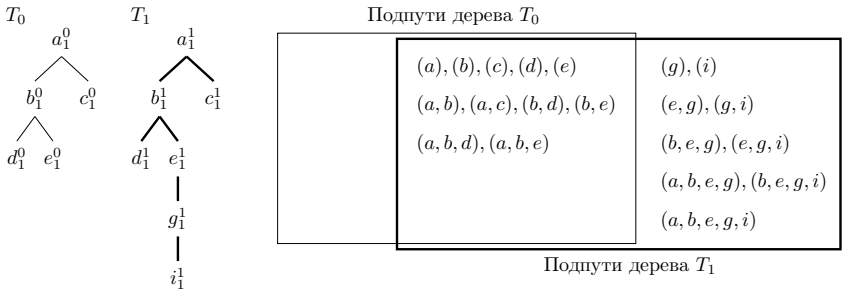

Рисунок 1.6. Подпути деревьев T_0 и T_2

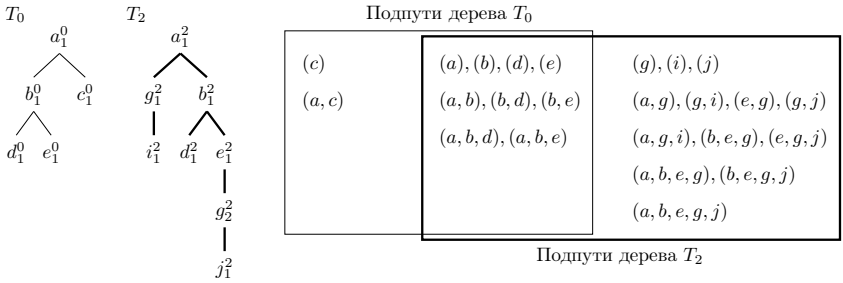

Рисунок 1.7. Подпути деревьев T_0 и T_2

На рис. 1.6 и рис. 1.7 можно увидеть набор общих подпутей между рассматриваемыми в качестве примера (рис. 1.1) деревом T_0 и T_1 и между деревом T_0 и T_2. Согласно этому методу, схожесть между деревом T_0 и деревьями T_1 и T_2 будет равна $S_{ss}(T_0, T_1) = 11$ и $S_{ss}(T_0, T_2) = 9$ соответственно.

Алгоритм по нахождению максимально схожего дерева дереву T_0 из набора деревьев TS можно ускорить, не вычисляя поочередно

схожесть каждого дерева из набора с запрашиваемым деревом. Для этого необходимо составить таблицу индексов $I[p]$ для корпуса деревьев, по которым будем искать схожесть с запрашиваемым деревом. Приведем пример и составим такую таблицу для корпуса деревьев T_1 и T_2 (рис. 1.1):

Таблица 1.10. Таблица индексов

p	$I[p]$	p	$I[p]$
a	$\{1,2\}$	$e \to g$	$\{1,2\}$
b	$\{1,2\}$	$g \to i$	$\{1,2\}$
c	$\{1\}$	$a \to g$	$\{2\}$
d	$\{1,2\}$	$g \to j$	$\{2\}$
e	$\{1,2\}$	$a \to b \to d$	$\{1,2\}$
g	$\{1,2\}$	$a \to b \to e$	$\{1,2\}$
i	$\{1,2\}$	$b \to e \to g$	$\{1,2\}$
j	$\{2\}$	$e \to g \to i$	$\{1\}$
$a \to b$	$\{1,2\}$	$a \to g \to i$	$\{2\}$
$a \to c$	$\{1\}$	$e \to g \to j$	$\{2\}$
$b \to d$	$\{1,2\}$	$a \to b \to e \to g$	$\{1,2\}$
$b \to e$	$\{1,2\}$	$b \to e \to g \to i$	$\{1\}$
		$b \to e \to g \to j$	$\{2\}$

В столбце p содержаться все подпути деревьев корпуса, в столбце $I[p]$ указывается список деревьев из корпуса, в которых есть данный подпуть.

Для каждого из подпутей запрашиваемого дерева, проходимся по столбцу p в поиске идентичного подпути. При совпадении, из правого столбца извлекаем информацию, в каком из деревьев из корпуса содержится данный подпуть.

При таком подходе объем занимаемой памяти при предварительной индексации корпусов деревьев составляет алгоритмическую сложность порядка $O(L*D^2)$, где L и D – максимальное количество листьев и максимальная глубина среди деревьев корпуса соответственно.

1.2.4. Метод, основанный на подсчете количества общих поддеревьев между деревьями

Впервые в работе [9] схожесть двух деревьев определяется количеством общих поддеревьев между ними (Tree Kernal). Понятие поддерева было введено в книге [10] и определяется следующим образом:

- Поддерево t состоит более чем из одного узла дерева T;
- узлы поддерева t должны содержать то же самое количество дочерних узлов что и узлы дерева T.

Поддеревья для рассматриваемых деревьев (рис. 1.1) T_0 и T_1 продемонстрированы на рис. 1.8. В силу большого количества поддеревьев дерева T_2, поддеревья между T_0 и T_2 продемонстрированы не будут.

Из рис. 1.8 можно сделать вывод, что дерево T_0 содержит 3 общих поддерева с деревом T_1. Количество общих поддеревьев между деревом T_0 и деревом T_2 равно 2. В этом случае схожесть между деревьями определяется как $S_{TK}(T_0, T_1) = 3$ и $S_{TK}(T_0, T_2) = 2$.

Для метода нахождения общих поддеревьев было разработано несколько алгоритмов. Самым большим недостатком первого из них (TK) — быстрое возрастание занимаемой памяти с количеством узлов в дереве из-за генерации огромного количества поддеревьев. Алгоритмы, разработанные позднее, позволили находить общие поддеревья, расходуя разумное количество памяти и не генерируя для каждого дерева все поддеревья. Одним из самых новых и эффективных можно считать (FTK). В табл. 1.11 приведена эффективность алгоритмов TK и FTK.

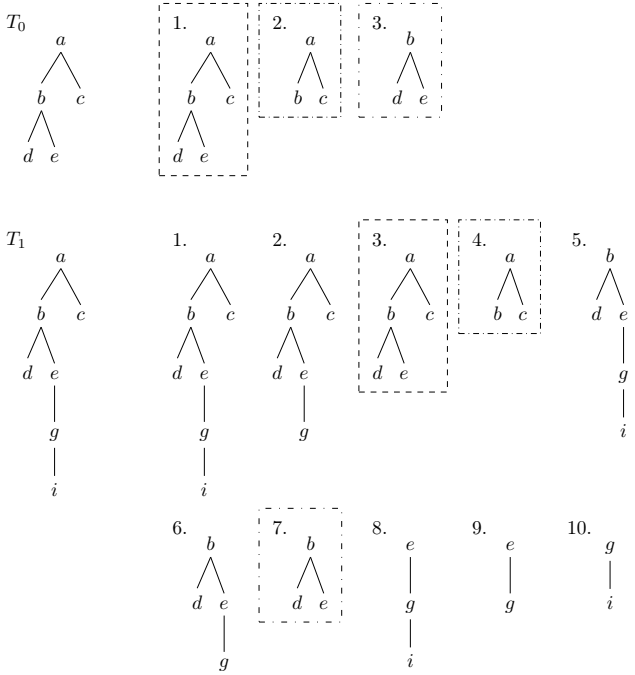

Рисунок 1.8. Поддеревья деревьев T_0 и T_1

Таблица 1.11. Сравнение алгоритмов FK и FTK, реализующий метод нахождения общх поддеревьев

Название алгоритма	Средняя скорость выполнения	Объем занимаемой памяти
TK	-	$O(2^n)$
FTK	$O(N_{T_1} + N_{T_2})$	$O(N_{T_1} * N_{T_2})$

1.2.5. Сравнение исследованных методов

Для каждого из исследованных методов определения схожести была подсчитана схожесть между деревом T_0 и деревьями корпуса, состоящего из двух деревьев — T_1 и T_2, рассматриваемыми в качестве

примера (рис. 1.1). Ниже приведена табл. 1.12, обобщающая полученные результаты.

Таблица 1.12. Сравнительная таблица схожестей между деревом T_0 и корпусом из деревьев T_1 и T_2 разными методами

Метод	Схожесть	Что определяет число схожести
Введение дистанции редактирования	$S_{TED}(T_0, T_1) = 2$ $S_{TED}(T_0, T_2) = 4$	кол–во операций, чтобы превратить одно дерево в другое
Наложение одного дерева на другое	$S_{TO}(T_0, T_1) = 2$ $S_{TO}(T_0, T_2) = 1$	кол–во узлов с одинаковыми правилами перехода при «наилучшем» наложении деревьев
Подсчет общих подпутей между деревьями	$S_{SS}(T_0, T_1) = 11$ $S_{SS}(T_0, T_2) = 9$	кол–во общих подпутей
Подсчет общих поддеревьев между деревьями	$S_{TK}(T_0, T_1) = 3$ $S_{TK}(T_0, T_2) = 2$	кол–во общих поддеревьев

Из табл. 1.12 можно сделать вывод о том, что каждый результат не нормирован, и их сравнение между собой вызывает определенные трудности. В дальнейшем необходимо вводить какую–либо метрическую оценку схожести на основе одного из представленных методов.

1.3. Текущие реализации систем по поиску схожих математических выражений на основе рассмотренных методов

Исследованные алгоритмы и методы, лежащие в их основе, по определению схожести упорядоченных деревьев нашли широкое при-

менение в области машинного обучения при анализе естественных языков. В ходе исследовательской части было найдено две поисковые системы, которые позволяют сравнивать MathML выражения между собой. Алгоритмы поиска, заложенные в их основу, применяют метод поиска общих подпутей и метод введения дистанции между двумя деревьями с помощью операций редактирования.

1.3.1. Поисковая система с алгоритмом нахождения общих подпутей

В работе [1] на основе алгоритма поиска схожих между деревьями подпутей, рассмотренного ранее в параграфе 1.2.3, была разработана система по поиску схожих выражений по заданному. Поскольку результатом применения алгоритма служит количество общих подпутей между деревом T_1 и T_2, то необходимо было ввести какую–либо метрическую оценку схожести. Авторы применяют коэффициент Жаккара:

$$K_J = \frac{S(T_1) \cap S(T_2)}{S(T_1) \cup S(T_2)} \tag{1.4}$$

В формуле 1.4 $S(T_i)$ — набор подпутей дерева T_i. В качестве корпуса авторы используют более 150 тысяч формул, собранных с сайта Wolfram [11], написанных в Presentation Markup, и с использованием Content Markup в качестве аннотаций.

В их работе представлены оценки по результативности поиска методом поиска общих подпутей. Результаты можно увидеть на рис. 1.9. Столбцы P и C содержат числа, указывающие, каким по счету по схожести было найдено ожидаемое выражение. В случае, если оно не было найдено в первых ста найденных выражениях, значение помечалось знаком x.

Авторы разработанной системы отметили, что при использовании данного алгоритма с выражениями, представленными в Presentation Markup, результаты были неудовлетворительными — одна из причин может быть в том, что функции и операции, в Presentation Markup обозначаются одним и тем же символом «*mo*», и при подсчете многих общих подпутей, содержащих данный символ, не учитывается эта особенность.

Запрос	*Ожидаемый результат*	P	C
$\sin(a+b)=\sin(a)\cos(b)+\cos(a)\sin(b)$	$\sin(a-b)=\sin(a)\cos(b)-\cos(a)\sin(b)$	x	6
$\int \sin z dz=-\cos z dz$	$\int \sin(az)dz=-\frac{\cos z}{a}$	x	39
$\int z e^{az}dz=\frac{e^{az}(-1+az)}{a^2}$	$\int z^3 e^{az}dz=\frac{e^{az}(-6+6az+3a^2z^2+a^3z^3)}{a^4}$	x	17
$\int (e^{cz})^v dz=\frac{(e^{cz})^v}{cv}$	$\int \sqrt{e^{cz}}dz=\frac{2\sqrt{e^{cz}}}{c}$	x	5
$\mathrm{ArcSin}(z)=\frac{3\pi}{4}-\frac{1}{2}\mathrm{Arctan}(\frac{1-2z^2}{2z\sqrt{1-z^2}})$	$\mathrm{ArcCos}(z)=-\frac{\pi}{4}+\frac{1}{2}\mathrm{Arctan}(\frac{1-2z^2}{2z\sqrt{1-z^2}})$	33	79

Рисунок 1.9. Результат поиска схожих математических выражений алгоритмом subpath для выражений в разметке Presentation Markup и Content Markup

1.3.2. EMERS

В работе [2] на основе метода введения дистанции между деревьями, рассмотренного ранее в параграфе 1.2.1, была разработана система EMERS по оценки качества распознавания математических выражений. Под распознанным математическим выражением понимаетсяы выражение в формате MathML, полученное в ходе распознавания выражения с изображения.

EMERS был разработан с учетом следующей особенности при распознавании математических нотаций — распознование символов может иметь разную степень сложности.

Рассмотрим два математических фрагмента: a^i и $e^{c^i_k}$. Оба из них содержит надстрочный индекс. Предположим, в результате распознавания этих фрагментов в первом случае система распознала один из них ai, а во втором — $e^{c^i k}$. Оба из них содержат одну ошибку, однако, не совсем корректно считать их равнозначными: можно сказать, что ошибка в первом случае более значима, чем во втором.

Для того, чтобы ввести разный уровень значимости в работе [12], были определены горизонтальные уровни для каждого символа. Базовый символ имеет значение горизонтального уровня равное 0. В зависимости от того, насколько высоко находится элемент относительно базового, ему присваивается значение горизонтального уровня равным $\{+1+2\cdots+n\}$ соответственно. Аналогично $\{-1-2\cdots-m\}$ для низлежащих элементов.

Было предложено давать штраф за ошибку в символе со значением горизонтального уровня k равное:

$$penalty = \frac{1}{|k| + 1} \tag{1.5}$$

К примеру, для выражения e^{c^i} значения горизонтального уровня для символов e, c и i будут равны $0, 1$ и 2. В случае допущения ошибки в символе c значение штрафа будет $\frac{1}{|1|+1} = 0.5$, а при ошибке в i штраф будет меньше и равен — $\frac{1}{|2|+1} = 0.33$.

EMERS подсчитывает штраф за неправильно распознанный символ аналогичным образом — чем выше или ниже горизонтальный уровень символа, тем ниже он будет, считая от корня дерева, в древовидном представлении выражения. EMERS за основу берет формулу штрафа 1.5. Ввиду того, что любой операции редактирования на основе алгоритма 1.2.1 можно назначить произвольный вес, он назначается следующим образом:

$$\gamma(a \to b) = \begin{cases} 0, & \text{если } a = b; \\ \frac{1}{L+1}, & \text{если } a \neq b. \end{cases} \tag{1.6}$$

, где $(a \to b)$ — операция изменения узла в дереве (операция замены, удаления, или вставки)

EMERS находит дистанцию между распознанным выражением и его правильным представлением по формуле 1.1, рассмотренной в параграфе 1.2.1. Ниже приведен пример сравнения двух выражений с помощью EMERS(рис. 1.10).

Actual expression: $e^{c^i_k}$

Recognized expression: $e^{c^{i^k}}$

```
Edit Cost is: 1.83333

The Modify operations are :
Mod- <msup> to <msubsup> on 6 and 14.
Mod- i to k on 9.
Mod- k to i on 11.

The Delete operations are :
Del- <mstyle> on 10 and 12.
Del- <msup> on 8 and 13.
```

Рисунок 1.10. Пример сравнения выражений с помощью EMERS

2. МОДИФИКАЦИЯ АЛГОРИТМА TREE OVERLAPPING ДЛЯ ПОИСКА СХОЖЕСТИ МЕЖДУ ВЫРАЖЕНИЯМИ В ФОРМАТЕ MATHML

2.1. Определение паттернов математической схожести

После рассмотрения видов разметки MathML в подразделе 1.1, в качестве разметки для определения синтаксической близости MathML выражений было решено взять за основу разметку Presentation Markup ввиду следующих соображений:

- многие элементы Content Markup отвечают за семантический смысл выражения или за отношения между его частями, что при отобрабражении не влияет на визуальное представление выражения, а следовательно и на синтаксическую структуру выражения;
- из-за богатого разнообразия элементов в Content Markup, некоторые операции, несущие разный семантический смысл, при отображении могут быть одинаковыми или очень похожими. В то же время Presentation Markup имеет фиксированный небольшой набор элементов, отвечающий за отображение конкретной конструкции выражения;
- в интернете наибольшее распространение получает Presentation Markup из-за того, что многие формулы, написанные на MathML, предназначены для визуального восприятия, а разметки Presentation Markup достаточно, чтобы визуально представить практически любое выражение необходимым образом.

Рассматриваемое MathML выражение решено представлять в виде упорядоченного ориентированного дерева E_1, с неограниченным количеством дочерних узлов. Всем узлам дерева v задать веса $w(v)$. В этом случае, весовая функция $w(T)$ дерева выражения или его части будет определяться как сумма весов всех узлов, которые в него

входят. Общая часть деревьев E_1 и E_2 — поддерево, полностью совпадающее с частью дерева E_1 и частью дерева E_2, а $E_1 \cap E_2$ — набор всех общих частей деревьев E_1 и E_2. Формула, определяющая схожесть выражений E_1 и E_2, определена как:

После рассмотрения методов определения схожести древовидных упорядоченных структур в разделе 1.2 и, при помощи работы [13], было решено определить синтаксическую схожесть математических выражений следующим образом:

$$Sim(E_1, E_2) = \frac{w(E_1 \cap E_2)}{w(E_1) + w(E_2)} \tag{2.1}$$

Также было решено ввести такой термин, как нормализация дерева выражения — это обезличивание переменных и числовых значений (терминальных узлов). Решено использовать следующие паттерны, часть из которых были заимствованы из работы [13], по возможной схожести мат. выражений:

- **Математическая эквивалентность**: E_1 и E_2 математически эквивалентны, если они семантически одинаковы (Пример: $\frac{d(sin(x))}{dx} = (sin(x))'$)
- **Идентичны**: E_1 и E_2 идентичны, если они абсолютны одинаковы.
- **Синтаксически идентичны**: E_1 и E_2 синтаксически идентичны, если идентичны после обезличивания переменных (Пример: $sin(a) = sin(b)$)
- **n–схожи**: нормализованные E_1 и E_2 n-схожи, если $sim(E_1, E_2) \geqslant n$, где n — параметрическое значение, определяющее порог схожести двух выражений.

    1. n–**схожесть подвыражений**: E_1 и E_2 n–схожи на уровне подвыражений, если у них существует общее поддерево, содержащее все терминальные узлы обоих деревьев (Пример: $six(x)^2$ и $\frac{sin(x)}{2}$, рис. 2.1)
    2. **Структурная** n–**схожесть**: E_1 и E_2 n–схожи и у них существует общая область, начинающаяся с корней обоих деревьев, из узлов с одинаковыми правилами перехода к последующим узлам. (Пример: $x + \sqrt{six(a)}$ и $x + \sqrt{2b}$, рис. 2.2)

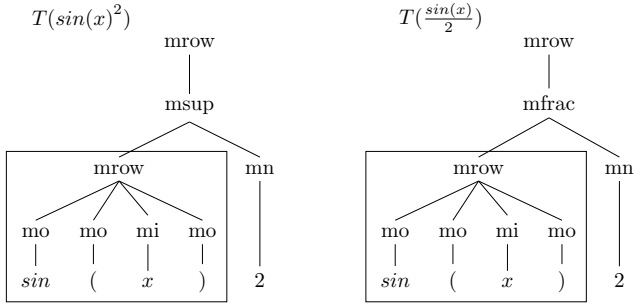

Рисунок 2.1. Структурная n–схожесть выражений $six(x)^2$ и $\frac{sin(x)}{2}$ представленных в формате MathML при $n \geqslant \frac{18}{26}$

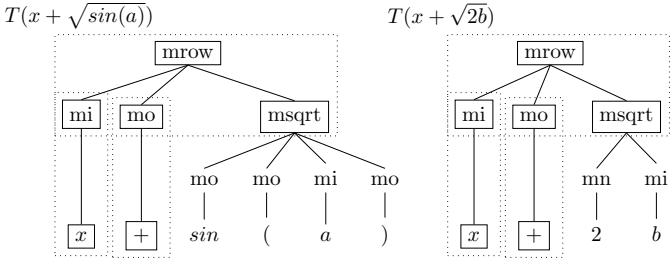

Рисунок 2.2. N–схожесть подвыражений выражений $x + \sqrt{six(a)}$ и $x + \sqrt{2b}$, представленных в формате MathML при $n \geqslant \frac{12}{24}$

На рис. 2.1 и рис. 2.2 в числителе неравенства — суммарное количество общих узлов между деревьями, а в знаменателе — суммарное количество всех узлов деревьев. Дробь формируется из формулы схожести математических выражений 2.1, представленной выше в этом же разделе.

В качестве одного из алгоритмов, рассмотренных в разделе 1 было решено взять алгоритм, в основе которого метод наложения одного дерева на другой, который определяет схожесть количеством узлов с одинаковыми правилами перехода между ними (параграф 1.2.2), так как он ближе всего подходит для вычисления синтаксической схожести выражений, представленных в формате MathML.

2.2. Модификация алгоритма

Алгоритм, выбранный за основу, не может быть использован без модификаций из-за того, что при расчете схожести деревьев между собой он:

- не учитывает одинаковые терминальные узлы;
- не учитывает одинаковые узлы, которые имеют одно и то же значение и которые стоят в том же самом порядке относительно своих родственных узлов, но не имеющих одинакового правила перехода от себя к другим узлам.

Для адаптации алгоритма для возможности поиска структурной схожести и схожести подвыражений математических выражений, представленных в формате MathML, решено было его модифицировать.

Прежде всего определим базовые понятия, неизменные для базового и модифицированного алгоритмов.

В ходе наложения деревьев друг на друга в узлах n_1 первого дерева и n_2 второго дерева, образуется набор пар узлов $L(n_1, n_2)$, которые также накладываются друг на друга. К примеру, при наложении дерева T_1 на дерево T_2 в узлах a^1 и a^2 на рис. 2.3 этот набор пар будет равен $L(a_1, a_2) = \{(a^1, a^2), (b^1, h^2), (c^1, c^2), (d^1, d^2), (e^1, j^2), (f^1, f^2), (g^1, g^2)\}$.

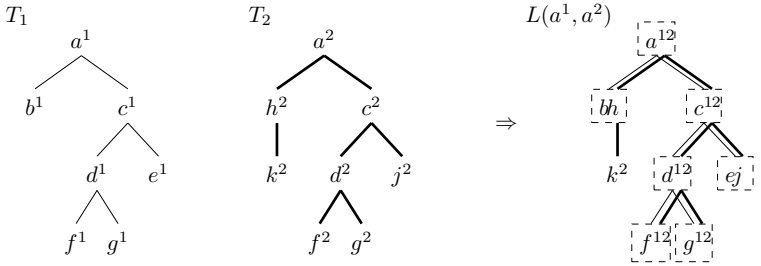

Рисунок 2.3. Пример наложения деревьев T_0 и T_1 в узлах a^1 и a^2

Пусть $ch(n, i)$ — i-ый дочерний узел узла n. Тогда множество пар узлов $L(n_1, n_2)$ формируется следующим образом:

1. $(n_1, n_2) \in L(n_1, n_2)$

2. Если $(m_1, m_2) \in L(n_1, n_2)$,
   то $(ch(m_1, i), ch(m_2, i)) \in L(n_1, n_2)$

3. $L(n_1, n_2)$ включает в себя все пары узлов, сгенерированные рекурсивно пунктом номер 2.

Также согласно базовому алгоритму — вводится понятие множества $N_{TO}(n_1, n_2)$. Оно определяется количеством нетерминальных пар узлов с одинаковыми правилами перехода и принадлежащими множеству накладываемых пар узлов при наложении одного дерева на другое в узлах n_1, n_2. $N_{TO}(n_1, n_2)$ выражается формулой 2.2:

$$N_{TO}(n_1, n_2) = \left\{ (m_1, m_2) \;\middle|\; \begin{array}{l} m_1 \in NT(T_1) \\ m_2 \in NT(T_2) \\ (m_1, m_2) \in L(n_1, n_2) \\ PR(m_1) = PR(m_2) \end{array} \right\} \tag{2.2}$$

Количество пар узлов $C_{TO}(n_1, n_2)$, содержащихся в множесте $N_{TO}(n_1, n_2)$, определяет количество общих узлов при наложении дерева T_1 на дерево T_2 в узла n_1 и n_2 и определяется формулой 2.3:

$$C_{TO}(n_1, n_2) = |N_{TO}(n_1, n_2)| \tag{2.3}$$

Для того, чтобы начать учитывать терминальные узлы, изменим часть условий для формирования множества $N_{TO}(n_1, n_2)$. В первую очередь, введем допущение, что m_1 и m_2 могут быть терминальными узлами. Пусть, если $NT(T)$ — все узлы дерева T за исключением терминальных, то $nodes(T)$ — все узлы дерева T.

Как известно, терминальные узлы не содержат дочерних узлов, ввиду этого у них нет правил перехода от них самих к дочерним узлам. Сделаем допущение: если узлы m_1 и m_2 деревьев T_1 и T_2 являются терминальным и их названия совпадают, то их правила перехода также совпадают. После внесения изменений, базовая формула 2.2, формирующая количество схожих пар узлов между деревьями при наложении их друг на друга в узлах n_1 и n_2 будет иметь вид:

$$N'_{TO}(n_1, n_2) = \left\{ (m_1, m_2) \;\middle|\; \begin{array}{l} m_1 \in \boldsymbol{nodes}(T_1) \\ m_2 \in \boldsymbol{nodes}(T_2) \\ (m_1, m_2) \in L(n_1, n_2) \\ \boldsymbol{PR}(m_1) = \boldsymbol{PR}(m_2) \end{array} \right\} \tag{2.4}$$

Подсчет количества общих узлов при наложении дерева T_1 на дерево T_2 в узла n_1 и n_2 в модифицированном алгоритме на этом этапе будет определятся схожим образом с формулой 2.3 и будет иметь вид:

$$C'_{TO}(n_1, n_2) = |N_{TO}(n_1, n_2)| \tag{2.5}$$

Пример подсчета схожести формулой 2.3 базового алгоритма и формулой 2.5 модифицированного алгоритма показан на рис. 2.4:

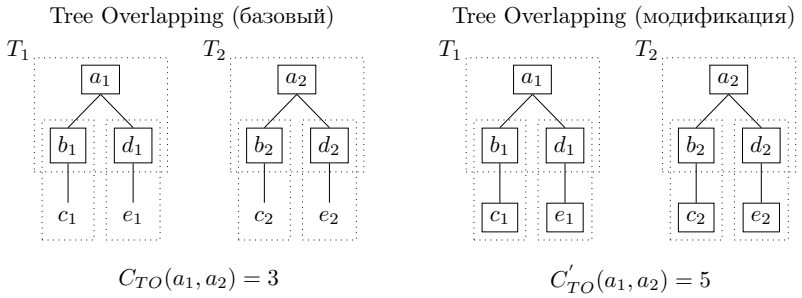

Рисунок 2.4. Пример подсчета схожести идентичных деревьев при их наложении в узлах a_1 и a_2 базовым алгоритмом (слева) и модифицированным (справа)

На примере, представленном на рис. 2.4, множества пар узлов N_{TO} будут определятся:

- $N_{TO}(a_1, a_2) = \{(a^1, a^2), (b^1, b^2), (d^1, d^2)\}$
- $N'_{TO}(a_1, a_2) = \{(a^1, a^2), (b^1, b^2), (d^1, d^2), (c^1, c^2), (e^1, e^2)\}$

Количества общих узлов между деревьями на рис. 2.4 будут равными:

- $C_{TO}(a_1, a_2) = |N_{TO}(a_1, a_2)| = 3$
- $C'_{TO}(a_1, a_2) = |N'_{TO}(a_1, a_2)| = 5$

Помимо схожести терминальных узлов, необходимо модифицировать алгоритм, чтобы он учитывал схожесть одинаковых узлов, которые имеют одно и то же значение и которые стоят в том же самом

порядке относительно своих родственных узлов, но не имеющих одинакового правила перехода от себя к другим узлам.

Введем понятия множества $P(n_1, n_2)$, которое содержит набор пар узлов в виде пути, начиная от узлов n_1, n_2, и продолжая до самых последних одинаковых узлов, стоящих среди своих братьев и сестер в одном и том же порядковом номере, и для которых узлы n_1 и n_2 являются потомками. $P(n_1, n_2)$ формируется следующим образом:

1. $(n_1, n_2) \in P$
2. Если $ch(parent(n_1), i) = ch(parent(n_2), i)$
   и $ch(parent(n_1), i) = n_1$
   и $ch(parent(n_2), i) = n_2$
   то $(parent(n_1), parent(n_2)) \in P$
3. $P(n_1, n_2)$ включает в себя все пары узлов, сгенерированные рекурсивно пунктом номер 2.

Введем понятие множества $P_{WPR}(n_1, n_2)$, которое определяется также, как и $P(n_1, n_2)$, с тем лишь исключением, что, если пара промежуточных узлов в пути содержит одно и то же правило перехода, то они в множество не заносятся. $P_{WPR}(n_1, n_2)$ формируется следующим образом:

1. $(n_1, n_2) \notin P_{WPR}$
2. Если $PR(parent(n_1)) \neq PR(parent(n_2))$
   и $ch(parent(n_1), i) = ch(parent(n_2), i)$
   и $ch(parent(n_1), i) = n_1$
   и $ch(parent(n_2), i) = n_2$
   то $(parent(n_1), parent(n_2)) \in P_{WPR}$
3. $P_{WPR}(n_1, n_2)$ включает в себя все пары узлов, сгенерированные рекурсивно пунктом номер 2.

На рис. 2.5 продемонстрирован пример формирования множества $P(n_1, n_2)$ и $P_{WPR}(n_1, n_2)$:

Как видно из рис. 2.5, множества P и P_{WPR} от узлов g_1 и g_2 будут равны:

- $P(g_1, g_2) = \{(g^1, g^2), (d^1, d^2), (c^1, c^2), (a^1, a^2)\}$

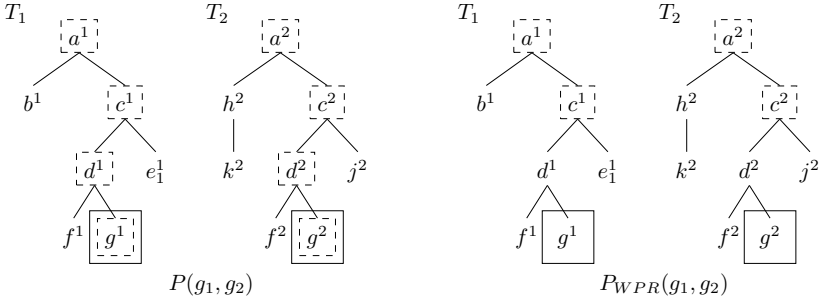

Рисунок 2.5. Определение множеств $P(n_1, n_2)$ (слева) и $P_{WPR}(n_1, n_2)$ (справа) на примере узлов g_1 и g_2 деревьев T_1 и T_2

- $P_{WPR}(g_1, g_2) = \{(c^1, c^2), (a^1, a^2)\}$

$P_{WPR}(g_1, g_2)$ не содержит пару узлов (d_1, d_2) из-за того, что узлы в ней имеют одинаковые правила перехода. По таким же рассуждениям пара узлов (g_1, g_2) также не заносится в множество $P_{WPR}(g_1, g_2)$.

Введем определение $top(n_1, n_2)$, обозначающее последнюю пару в множестве $P(n_1, n_2)$ или в множестве $P_{WPR}(n_1, n_2)$:

$$top(n_1, n_2) = p_{last}(n_1, n_2), \qquad p_{last} \in P \vee P_{WPR} \tag{2.6}$$

На рис. 2.5 $top(g_1, g_2)$ будет определятся одинаково для $P(d_1, d_2)$ и для $P_{WPR}(d_1, d_2)$ и равняться в обоих случаях $top(d_1, d_2) = (a_1, a_2)$.

Введем формальное определение множества пар узлов в виде пути до узлов m_1, m_2, получившихся в результате наложения узла n_1 на узел n_2 деревьев T_1 и T_2:

$$P_{TO}(n_1, n_2) = \left\{ (m_1, m_2) \;\middle|\; \begin{array}{l} (p_1, p_2) \in N_{TO}(n_1, n_2) \\ (m_1, m_2) \in P_{WPR}(p_1, p_2), \\ \qquad \text{если} \quad top(m_1, m_2) = (n_1, n_2) \end{array} \right\} \tag{2.7}$$

Окончательно преобразуем формулу подсчета схожести деревьев T_1 и T_1 при наложение узлов n_1 и n_2. С учетом формулы 2.7 формула C_{TO} будет иметь вид:

$$C''_{TO}(n_1, n_2) = |N'_{TO}(n_1, n_2)| + |P_{TO}(n_1, n_2)| \tag{2.8}$$

Пример подсчета схожести деревьев T_0 и T_1 в узлах n_1 и n_2 базовым и модифицированным алгоритмами приведен на рис. 2.6:

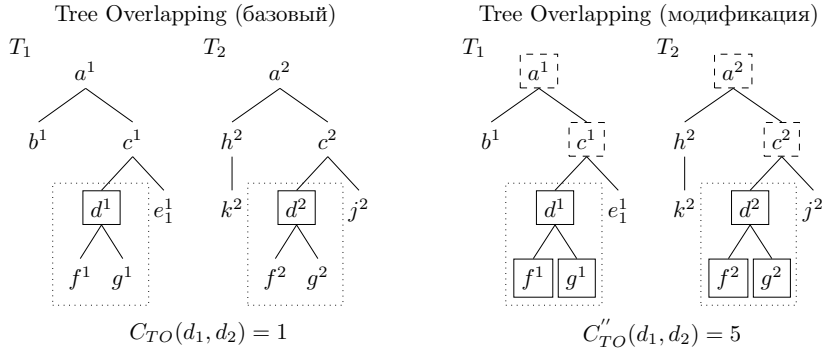

Рисунок 2.6. Пример поиска схожести между деревьями T_0 и T_1 базовым алгоритмом (слева) и модифицированным алгоритмом (справа)

Формула схожести между деревом T_1 и T_2 после модификации определяется следующим образом:

$$S'_{TO}(T_1, T_2) = \max_{n_1 \in nodes(T_1), n_2 \in nodes(T_2)} C''_{TO}(n_1, n_2) \tag{2.9}$$

В дальнейшем будут использоваться определения модифицированного алгоритма и для простоты восприятия формул $C''_{TO}(n_1, n_2)$ и $S'_{TO}(T_1, T_2)$ они будут цитироваться как $C_{TO}(n_1, n_2)$ и $S_{TO}(T_1, T_2)$ несмотря на то, что эти формулы соответствуют обозначениям формул базового алгоритма.

Алгоритм создания таблицы индексов при поиске схожести дерева по корпусу деревьев также претерпел изменения. Теперь таблица индексов может содержать терминальные правила перехода с терминальными узлами. Элементы таблицы индексов определяются следующей формулой:

$$I[p] = \left\{ (m) \;\middle|\; \begin{array}{l} T \in F \\ m \in \boldsymbol{nodes}(T) \\ p = \boldsymbol{PR}(m) \end{array} \right\} \tag{2.10}$$

Приведем пример составления таблиц индексов для корпуса из деревьев T_1 и T_2 (рис. 2.7) по базовому алгоритму (табл. 2.1) и по

модифицированному (табл. 2.2):

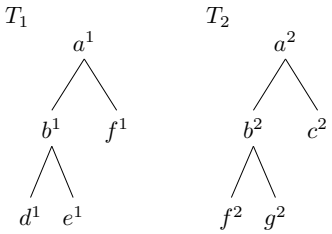

Рисунок 2.7. Корпус из деревьев T_1 и T_2

Таблица 2.1. Таблица индексов T_1, T_2 (базовый алгоритм)

p	$I[p]$
$a \to b\, f$	a^1
$b \to d\, e$	b^1
$a \to b\, c$	a^2
$b \to f\, g$	b^2

Таблица 2.2. Таблица индексов T_1,T_2 (модифицированный алгоритм)

p	$I[p]$
$a \to b\, f$	a^1
$b \to d\, e$	b^1
f	f^1, f^2
d	d^1
e	e^1
$a \to b\, c$	a^2
$b \to f\, g$	b^2
c	c^2
g	g^2

Как видно из табл. 2.1 и табл. 2.2, они отличаются друг от друга лишь наличием «пустых» правил перехода и терминальных узлов в таблице индексов.

В этом разделе, кроме вышеописанных, появляются некоторые дополнительные изменения при поиске схожести подвыражений. Эти изменения будут подробно описаны в разделе 2.4, посвященному поиску схожести подвыражений.

Далее, в подразделах 2.3 и 2.4 будут приведены примеры нахождения структурной схожести и схожести подвыражений модифициро-

ванным алгоритмом.

2.3. Применение алгоритма при вычислении структурной схожести

Приведем пример поиска структурной схожести между деревом T_0 и корпусом деревьев T_1 и T_2 (рис. 2.8). Для этого составим таблицу индексов (табл. 2.3) для дерева T_0 и таблицу индексов (табл. 2.4) для корпуса деревьев T_1 и T_2.

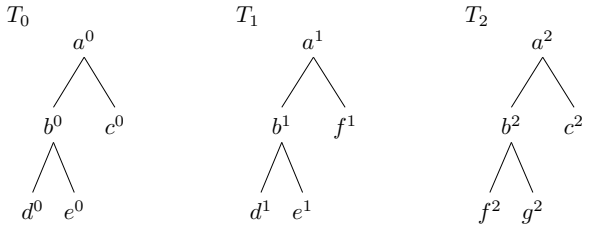

Рисунок 2.8. Запрашиваемое дерево T_0 и корпус из деревьев T_1 и T_2

Таблица 2.3. Таблица индексов T_0

p	$I[p]$
$a \to b\,c$	a^0
$b \to d\,e$	b^0
c	c^0
d	d^0
e	e^0

Таблица 2.4. Таблица индексов T_1,T_2

p	$I[p]$
$a \to b\,f$	a^1
$b \to d\,e$	b^1
f	f^1, f^2
d	d^1
e	e^1
$a \to b\,c$	a^2
$b \to f\,g$	b^2
c	c^2
g	g^2

Табл. 2.3 и табл. 2.4 сформированы по формуле 2.10. Левый столбец p содержит информацию о правиле перехода, который присутству-

ет среди узлов деревьев корпуса, а правый столбец $I[p]$ информацию о том, какой узел содержит данное правило перехода. По полученным таблицам индексов (табл. 2.3 и табл. 2.4), составим табл. 2.5 между узлами дерева T_0 и деревьями корпуса T_1 и T_2.

Таблица 2.5. Таблица Top между узлами дерева T_0 и узлами деревьев T_1 и T_2

(n, m)	$top(n, m)$
(a^0, a^2)	(a^0, a^2)
(b^0, b^1)	(a^0, a^1)
(c^0, c^2)	(a^0, a^2)
(d^0, d^1)	(a^0, a^1)
(e^0, e^1)	(a^0, a^1)

У табл. 2.5 левый столбец (n, m) содержит совпавшие узел дерева T_0 и узел дерева корпуса с одинаковыми правилами перехода. Правый столбец $top(n, m)$ содержит пару узлов, определенную формулой 2.6.

Подсчитываем количество одинаковых пар узлов из правого столбца табл. 2.5, являющегося множеством, определенном по формуле 2.4 и получаем $|N_{TO}(a_0, a_1)| = 3$, $|N_{TO}(a_0, a_2)| = 2$. С учетом того, что пара узлов (a_0, a_1) принадлежит множеству P_{WPR}, определенному ранее в разделе 2.2, получаем $|P_{TO}(a_0, a_1)| = 1$.

Согласно формуле 2.9 схожесть между запрашиваемым деревом T_0 и деревом корпуса T_1 будет равна $S_{TO}(T_0, T_1) = 3 + 1 = 4$, а между запрашиваемым деревом T_0 и деревом корпуса T_2 равна $S_{TO}(T_0, T_2) = 2$. Схожесть между деревьями продемонстрирована на рис. 2.9.

2.4. Применение алгоритма при вычислении схожести подвыражений

Приведем пример поиска схожести подвыражений между деревом T_0 и корпусом из деревьев T_1 и T_2 (рис. 2.10). Для этого составим таблицу индексов (табл. 2.6) для дерева T_0 и таблицу индексов (табл. 2.7) для корпуса T_1 и T_2.

Табл. 2.6 и табл. 2.7 сформированы по формуле 2.10. Левый столбец p содержит информацию о правиле перехода, который присутству-

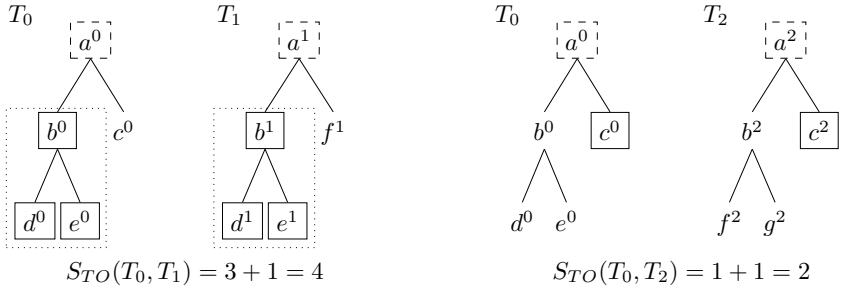

Рисунок 2.9. Структурная схожесть между деревьями T_0 и T_1 (слева) и между деревьями T_0 и T_2 (справа)

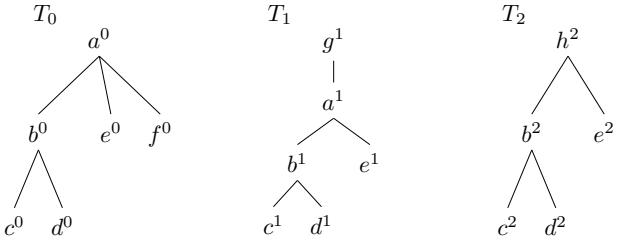

Рисунок 2.10. Запрашиваемое дерево T_0 и корпус из деревьев T_1 и T_2

ет среди узлов деревьев корпуса, а правый столбец $I[p]$ информацию о том, какой узел содержит данное правило перехода. По полученным таблицам индексов (табл. 2.6 и табл. 2.7), составим табл. 2.8 между узлами дерева T_0 и деревьями корпуса T_1 и T_2.

У табл. 2.8 левый столбец (n, m) содержит совпавшие узел дерева T_0 и узел дерева корпуса с одинаковыми правилами перехода. Правый столбец $top(n, m)$ содержит пару узлов, определенную формулой 2.6. Далее, подсчитываем количества одинаковых пар узлов из правого столбца табл. 2.5, являющегося множеством, определенном по формуле 2.4 и получаем:

- $|N_{TO}(a_0, a_1)| = 4$
- $|N_{TO}(b_0, b_2)| = 3$

С учетом того, что пара узлов (a_0, a_1) принадлежит множеству

Таблица 2.6. Таблица индексов T_0

p	$I[p]$
$a \to b\,c\,f$	a^0
$b \to c\,d$	b^0
e	e^0
f	f^0
c	c^0
d	d^0

Таблица 2.7. Таблица индексов T_1,T_2

p	$I[p]$
$g \to a$	g^1
$a \to b\,e$	a^1
$b \to c\,d$	b^1, b^2
e	e^1
c	c^1, c^2
d	d^1, d^2
$h \to b$	h^2
$b \to f\,g$	b^2

Таблица 2.8. Таблица Top между узлами дерева T_0 и узлами деревьев T_1 и T_2

(n, m)	$top(n, m)$
(b^0, b^1)	(a^0, a^1)
(b^0, b^2)	(b^0, b^2)
(e^0, e^1)	(a^0, a^1)
(c^0, c^1)	(a^0, a^1)
(c^0, c^2)	(b^0, b^2)
(d^0, d^1)	(a^0, a^1)
(d^0, d^2)	(b^0, b^2)

P_{WPR}, определенному ранее в разделе 2.2, то $|P_{TO}(a_0, a_1)| = 1$.

Схожести, полученные в результате наложения дерева T_0 на деревья корпуса T_1 и T_2 в узла a_0, a_1 и b_0, b_2 определяются формулой 2.8 и равны:

- $CTO(a_0, a_1) = |N_{TO}(a_0, a_1)| + |P_{TO}(a_0, a_1)| = 5$
- $CTO(b_0, b_2) = |N_{TO}(a_0, a_1)| = 3$

Далее необходимо узнать, являются ли полученные схожие области поддеревьями. При наложении узлов n_1 и n_2 друг на друга, мы

будем считать, что, в случае, если полученная область является поддеревом, то она должна содержать количество листьев равное большему из двух количеств листьев поддеревьев, образованных отдельно от узлов n_1 и n_2.

К примеру, на рис. 2.10 поддерево дерева T_0, образованное от узла a_0 содержит 4 терминальных узла, а поддерево T_1, образованное от узла a_1 содержит 3 терминальных узла. При наложении деревьев в узлах a_0 и a_1, количество листьев, которое должно быть покрыто «потенциальным получившимся поддеревом» равно $\max\{3, 4\} = 4$.

Введем понятие $Leaves_{cover}(n_1, n_2)$, которое будет содержать два числа — первое будет определять, какое общее количество листьев покроется при наложении узлов n_1, n_2 друг на друга, а второе будет показывать сколько необходимо покрыть листьев, чтобы полученная при таком наложении область была поддеревом.

Для нашего рассматриваемого примера, еще раз пройдем таблицу Top (табл. 2.8) и просмотрим наличие терминальных пар узлов из левого столбца у узлов (n_1, n_2) из правого, и если они терминальные, прибавим к значению $Leaves_{cover}(n_1, n_2)$ единицу. Получим следующие значения:

- $Leaves_{cover}(a_0, a_1) = 3/4$
- $Leaves_{cover}(b_0, b_2) = 3/3$

Можно сказать, что область, полученная при наложении узлов a_0 и a_1 не является поддеревом, а область, полученная при наложении b_0 и b_2 — является. Это продемонстрировано на рис. 2.11.

На рис. 2.11 видно, что схожесть между деревьями T_0 и T_2 в узлах b_0, b_2 полностью покрыта поддеревом, в то время как схожесть между T_0 и T_1 в узлах a_0, a_1 нет. На текущем этапе задача по поиску схожести подвыражений между деревьями T_0 и T_1 не решена. Из-за этого необходимо еще раз повторить алгоритм нахождения схожести, преобразовав деревья T_0 и T_1 к деревьям $T_0^{'}$ и $T_1^{'}$, которые будут отличатся вычеркнутыми узлами, которые при наложении не покрыли все поддерево. Кроме этого, вычеркиваются и все вышележащие относительно них узлы. Преобразование деревьев T_0, T_1 к $T_0^{'}$ и $T_1^{'}$ продемонстрировано на рис. 2.12.

Еще раз составим таблицу индексов, но теперь уже только для дерева $T_0^{'}$ (табл. 2.9) и $T_1^{'}$ (табл. 2.10):

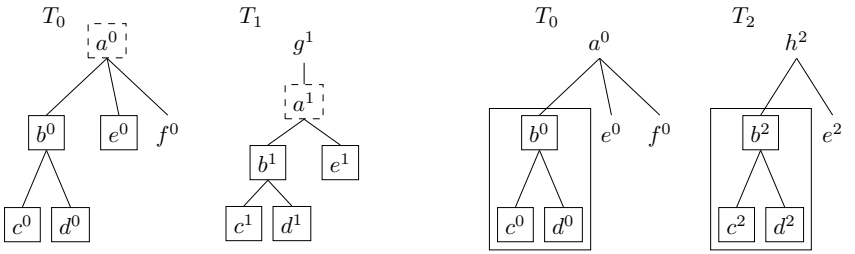

Рисунок 2.11. Схожесть между деревьями T_0 и узлами корпуса в узлах a_0, a_1 и b_0, b_2

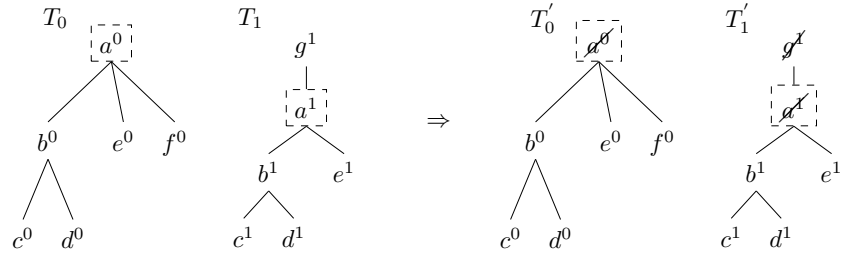

Рисунок 2.12. Переход от деревьев T_0, T_1 к $T_0^{'}$, $T_1^{'}$ методом вычеркивания a_0, a_1 и всех вышележащих отнотилеьно них узлов

По табл.2.11 подсчитываем количество одинаковых пар узлов из правого столбца и получаем:

- $|N_{TO}(b_0, b_1)| = 3$
- $|N_{TO}(e_0, e_1)| = 1$

В результате наложения дерева $T_0^{'}$ на дерево $T_1^{'}$ в узлах b_0, b_1 и e_0, e_1 схожести будут равны:

- $CTO(b_0, b_1) = |N_{TO}(b_0, b_1)| = 3$
- $CTO(e_0, e_1) = |N_{TO}(e_0, e_1)| = 1$

Таблица 2.9. Таблица индексов $T_0^{'}$

p	$I[p]$
$b \to c\,d$	b^0
e	e^0
f	f^0
c	c^0
d	d^0

Таблица 2.10. Таблица индексов $T_1^{'}$

p	$I[p]$
$b \to c\,d$	b^1
e	e^1
c	c^1
d	d^1

Таблица 2.11. Таблица Top между узлами дерева $T_0^{'}$ и узлами дерева $T_1^{'}$

(n, m)	$top(n, m)$
(b^0, b^1)	(b^0, b^1)
(e^0, e^1)	(e^0, e^1)
(c^0, c^1)	(b^0, b^1)
(d^0, d^1)	(b^0, b^1)

Подсчитываем количество листьев, которые покрыли области при наложении узлов b_0, b_1 и e_0, e_1:

- $Leaves_{cover}(b_0, b_1) = 2/2$
- $Leaves_{cover}(e_0, e_1) = 1/1$

Поскольку в обоих случаях были покрыты все листья, то обе области, полученные в результате наложения b_0, b_1 и e_0, e_1, являются поддеревьями и нам подходят. В случае, если бы это было не так, нам бы пришлось еще раз переходить от рассмотрения деревьев $T_0^{'}$ и $T_1^{'}$ к последующим, и так до тех пор, пока все области, полученные при дальнейшем наложении друг на друга в одинаковых узлах не стали бы являться поддеревьями.

Вернемся к исходной задаче и подсчитаем все схожести, полученные при наложении одинаковых узлов между запрашиваемым деревом T_0 и деревьями корпуса T_1 и T_2, которые покрыты поддеревом:

- $CTO(b_0, b_2) = 3$

- $CTO(b_0, b_1) = 3$
- $CTO(e_0, e_1) = 1$

Согласно формуле 2.9 схожесть между запрашиваемым деревом T_0 и деревом корпуса T_1 будет равна $S_{TO}(T_0, T_1) = 3$, а между запрашиваемым деревом T_0 и деревом корпуса T_2 также равна $S_{TO}(T_0, T_2) = 3$. Схожесть между деревьями продемонстрирована на рис. 2.13.

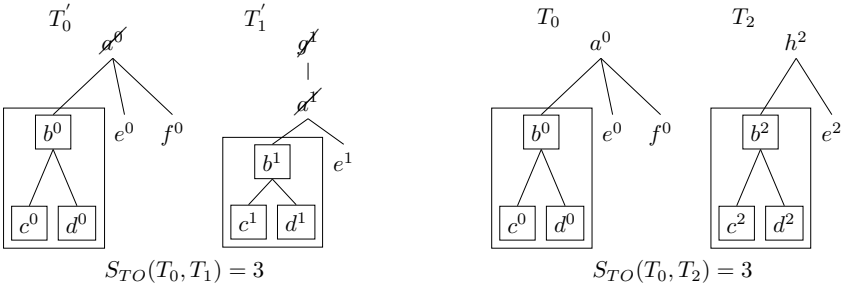

Рисунок 2.13. Схожесть подвыражений между деревьями T_0 и T_1 (слева) и между деревьями T_0 и T_2 (справа)

3. ОПИСАНИЕ ПРИЛОЖЕНИЯ, РЕАЛИЗУЮЩЕГО ПОИСК СИНТАКСИЧЕСКИ СХОЖИХ МАТЕМАТИЧЕСКИХ ВЫРАЖЕНИЙ НА ОСНОВЕ РАЗРАБОТАННОГО АЛГОРИТМА

С целью демонстрации поиска и тестирования математических выражений модифицированным алгоритмом было разработано приложение–прототип, которое реализует следующие заявленные функциональные требования:

- возможность парсинга математических выражений в формате MathML;
- хранение древовидных представлений математических выражений в базе;
- реализация поиска схожих математических выражений из базы заданному математическому выражению;
    1. на основе структурной схожести;
    2. на основе схожести подвыражений;
- возможность визуального представления полученных результатов.

Для реализации было решено выбрать язык Java в связи с наличием библиотеки по визуализации выражений в формате MathML, а также опытом разработки программ на этом языке. В качестве среды разработки была выбрана IDE Netbeans.

На рис. 3.1 приведена диаграмма основных классов разработанного приложения. Для большей наглядности на диаграмме отражены только основные методы классов. Описание разработанных классов будет приведено в последующих параграфах.

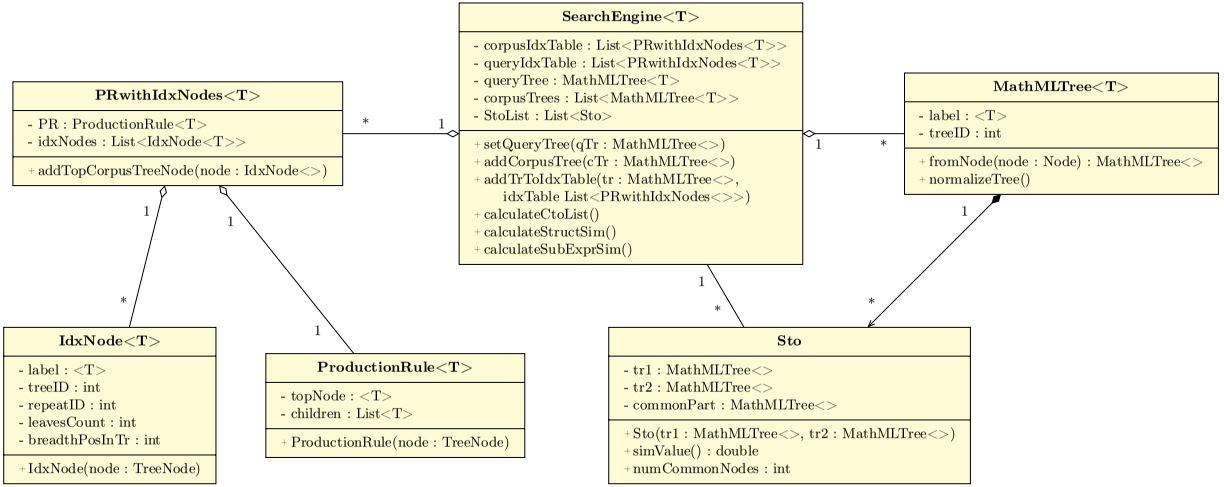

Рисунок 3.1. Диаграмма UML основных классов приложения (часть методов опущена для большей наглядности)

3.1. Генерация MathML выражений и их обработка

Все файлы с математическими выражениями в формате MathML хранятся отдельно в каждом файле с расширением «.xml». Их генерация производилась путем написания необходимых выражений в формате LaTeX и дальнейшей их конвертации с помощью конвертера [14].

Для парсинга файлов с математическими выражениями используется соответствующий метод класса MathMLParserSupport из сторонней библиотеки net.sourceforge.jeuclid. Полученный объект класса Document принимается параметром для метода, который генерирует объект разработанного класса MathMLTree для последующей работы с древовидным представлением выражения. Ниже приведен фрагмент кода, иллюстрирующий данный порядок действий:

```
1 // загружаем xml файл
2 File f = new File(path, fileName);
3
4 // парсим файл
5 Document domDocument = MathMLParserSupport.parseFile(f);
6
7 // получаем объект для работы с древовидным представление выражения
8 this.mathMLTree = MathMLTree.fromNode(domDocument.getDocumentElement())
      ;
```

Поскольку данное приложение носит цель демонстрации алгоритма, все данные о математических выражениях находятся отдельно в каждом файле, а не в базе данных. Они загружаются и обрабатываются данным фрагментом кода каждый раз во время исполнения программы.

3.2. Сущностные классы

3.2.1. Класс, отвечающий за хранение и работу с древовидным представлениям выражения

Для хранения древовидных представлений выражений был разработан класс MathMLTree (прилож. А.4), который наследовал реализацию класса DefaultMutableTreeNode, определенного в пакете javax.swing.tree.TreeNode, который содержит в себе два основных поля: объекта родителя и список объектов дочерних узлов, принадлежащих тому же самому классу, и ряд методов для работы с узлами.

Решение наследоваться было принято ввиду наличия всех основных методов, необходимых для работы с древовидным представлени-

ем объекта. Из них можно выделить следующие:

- **public int getChildCount()** : возвращает количество дочерних узлов;
- **public boolean isRoot()** : определяет, является ли данный узел корневым, т.е. не имеет вышележащих узлов;
- **public Enumeration breadthFirstEnumeration()**: возвращает список узлов в порядке обхода узлов дерева в ширину (рис. 3.2);
- **public Enumeration depthFirstEnumeration()**: возвращает список узлов в порядке обхода узлов дерева в глубину (рис. 3.2);

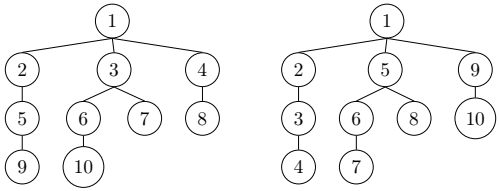

Рисунок 3.2. Пример обхода дерева в ширину (слева) и в глубину (справа)

- **public int getSiblingCount()** : возвращает количество дочерних узлов от родителя текущего узла;
- **public DefaultMutableTreeNode getNextSibling()** : выдает следующего узла–родственника по списку;
- **public boolean isLeaf()**: определяется, имеет ли данный узел дочерние узлы или нет.

Кроме вышеперечисленных методов, унаследованных от класса DefaultMutableTreeNode, были также добавлены поле–идентификатор узла и следующие основные методы:

- **public static MathMLTree fromNode(Node node)**: возвращает дерево выражения из древовидной структуры узла типа Node (получаемой после парсинга xml документа);

- **public static MathMLTree fromString(String s)**: возвращает дерево выражения из строки (данный метод использовался для генерации тестовых деревьев);
- **public void normilizeTree()**: нормализует дерево выражения, обезличивая (замена на обобщающие названия) необходимые узлы;
- **public int getSiblingsNumber()**: возвращает количество дочерних узлов от родителя текущего узла;
- **public int getLabelIdxTime()**: возвращает каким по счету встретился этот узел среди узлов с таким же названием от корня дерева при обходе дерева в ширину;
- **public int getNodeBreadthOrder()**: возвращает номер узла при обходе дерева в ширину, начиная от корневого узла.
- **public TreeNode getNodeByBreadthOrderIndex(int n)**: возвращает узел под номером n при обходе поддерева в ширину, начинающегося от текущего узла.

Представленные методы были добавлены для последующего их использования в ходе поиска необходимой информации в древовидных представлениях выражений во время поиска их схожести по разработанному алгоритму.

3.2.2. Классы, отвечающие за составление таблиц индексации деревьев выражений

Для хранения дополнительной информации об узлах был разработан класс IdxNode (прилож. А.2). Класс содержит такие поля как:

- идентификатор узла (название)
- идентификатор дерева, в котором находится данный узел
- каким по счету встречается узел с тем же идентификатором в дереве
- количество листьев у узла
- позиция в дереве при обходе в ширину.

Кроме этого был разработан метод для гибкой оценки равенства объектов данного класса. В зависимости от параметра, объекты признаются равными, если совпадают все поля объектов или же некоторая часть из них.

Для хранения правил перехода от одного узла к последующим был разработан класс ProductionRule (прилож. А.5). При этом важно отметить, что данный класс содержит лишь информацию об идентификаторе (названии) узлов и не привязывается к конкретным узлам в дереве.

Для связывания информации между правилами перехода и узлами, которые содержат данное правило, был разработан класс PRwithIdxNodes (прилож. А.6), который агрегирует в себя объект класса ProductionRule и список объектов IdxNode.

С помощью классов, описанных в данном параграфе составляются таблицы индексов (список из правил перехода, каждый из которых содержит набор узлов с данным правилом перехода) для узлов математических выражений в древовидном представлении.

3.2.3. Класс, реализующий поиск схожести по разработанному алгоритму

Для реализации поиска структурной схожести и схожести подвыражений был разработан класс SearchEngine (прилож. А.1).

Ниже приведены поля класса с их описанием:

- **MathMLTree qTr**: дерево выражения, для которого будет производиться поиск схожести;
- **List<MathMLTree> cTrs**: список из корпуса деревьев выражений, по которым будет производиться поиск;
- **List<PRwithIdxNodes> qIdxTb**: таблица индексов для запрашиваемого дерева;
- **List<PRwithIdxNodes> cIdxTb**: таблица индексов для корпуса деревьев;
- **ArrayList<Sto> stoList**: список объектов, содержащих информацию о схожести между запрашиваемым деревом *qTr* и каждым деревом корпуса *cTrs*.

Ниже приведены основные методы данного класса с их описанием:

- **public void setQueryTree(MathMLTree qTr)**: задаем дерево выражения, для которого будет производится поиск схожести (во время выполнения метода данное дерево индексируется для заполнения таблицы индексов запрашиваемого дерева *qIdxTb*);

- **public void addCorpusTree(MathMLTree cTr)**: добавляем дерево выражения в корпус, по которому будет производиться поиск (во время выполнения метода данное дерево индексируется для дополнения информации к таблице индексов корпуса *cIdxTb*);

- **public void calculateStructuralSimilarity()**: производится подсчет схожести подвыражений между запрашиваемым деревом *qTr* и каждым деревом корпуса *cTrs* (алгоритм подробно изложен в подразделе 2.3), результат заносится в *stoList*;

- **public void calculateSubExprSimilarity()**: производится подсчет структурной схожести между запрашиваемым деревом *qTr* и каждым деревом корпуса *cTrs* (алгоритм подробно изложен в подразделе 2.4), результат заносится в *stoList*;

В следующем фрагменте кода приведен пример использования объекта данного класса для подсчета одного из видов схожести между выражениями, хранимыми в файлах, представленных в формате MathML:

```
1  MathExpression expr1 = new MathExpression(PATH_TO_MATHML_DATABASE, "
       expr1.xml");
2  MathExpression expr2 = new MathExpression(PATH_TO_MATHML_DATABASE, "
       expr2.xml");
3  MathExpression expr3 = new MathExpression(PATH_TO_MATHML_DATABASE, "
       expr3.xml");
4
5  SearchEngine se = new SearchEngine();
6  se.setQueryTree(expr1.getMathMLTree());
7  se.addCorpusTree(expr2.getMathMLTree());
8  se.addCorpusTree(expr3.getMathMLTree());
9
10 se.calculateSubExprSimilarity();
```

3.2.4. Класс, хранящий информацию о схожести между выражениями

Для хранения информации о схожести между древовидными представлениями математических выражений, был разработан класс Sto (прилож. А.7). Класс содержит:

- ссылки на объекты сравниваемых деревьев выражений;
- вид схожести между деревьями выражений;
- общую область из узлов, определяющую схожесть между деревьями выражений.

С помощью объекта данного класса можно получить информацию о количестве схожих узлов между деревьями, о месте начала схожести и т.п.

3.3. Визуальное представление результатов

Для настройки параметров поиска и представления результатов была использована библиотека Swing для создания графического интерфейса. На рис. 3.3 и рис. 3.4 приведены окна графического интерфейса, отвечающие за установку параметров поиска и вывод результатов.

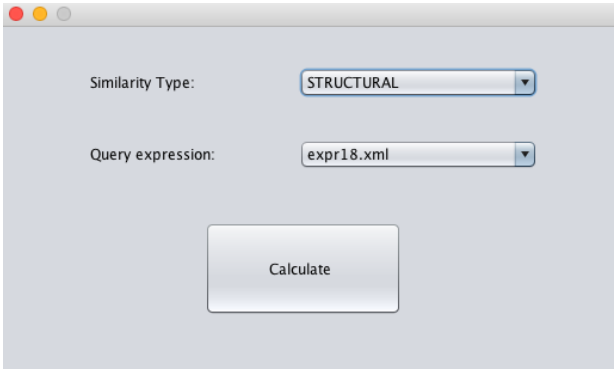

Рисунок 3.3. Установка параметров поиска в графическом интерфейсе (SettingsFrame)

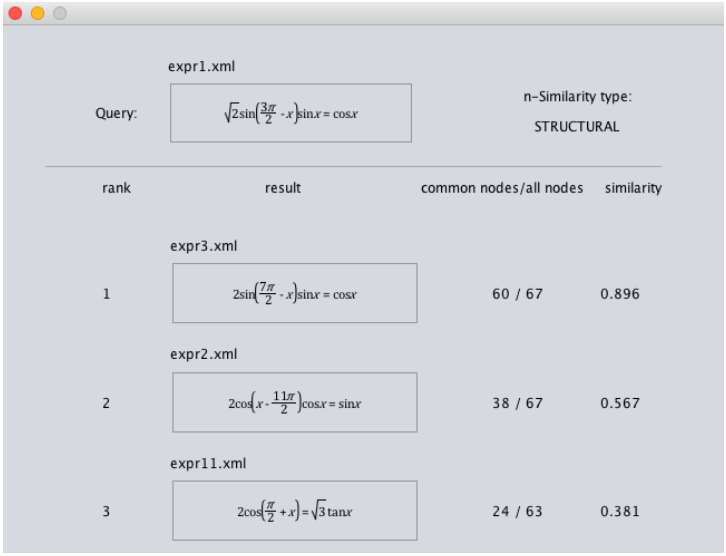

Рисунок 3.4. Вывод результата поиска схожести заданному выражению в графическом интерфейсе (ResultFrame)

Для отображения математических выражений используется библиотека net.sourceforge.jeuclid, которая содержит класс, отвечающий за отображение математического выражения средствами Swing (javax.swing.JComponent).

```
Running searchengine.SearchEngineTest
Sto(T0,T1) = 16 / 67 (mfrac,mfrac)
Sto(T0,T2) = 4 / 58 (mn,mn)
Sto(T0,T3) = 4 / 58 (mn,mn)
Sto(T0,T4) = 4 / 60 (mn,mn)
Sto(T0,T5) = 6 / 63 (msqrt,msqrt)
Sto(T0,T6) = 4 / 64 (mn,mn)
```

Рисунок 3.5. Консольный вывод результата поиска схожести заданному выражению

Графический интерфейс, предоставляет не всю информацию относительно результатов поиска — он показывает лишь пять выражений, оказавшихся самыми схожими с заданным. Ввиду этого, кроме графического интерфейса, также присутствует возможность консоль-

ного вывода информации. На рис. 3.5 приведен пример консольного вывода информации относительно схожести каждого уравнения из базы с заданным после поиска схожести подвыражений между ними.

Данные способы визуального представления результатов поиска позволяют полноценно протестировать эффективность алгоритма. Тестирование алгоритма и оценка его эффективности будет произведена в разделе 4.

4. ОЦЕНКА ЭФФЕКТИВНОСТИ АЛГОРИТМА

Существует проблема в виде отсутствия общепринятого корпуса математических выражений, разбитых по определенным классам схожести. Это вызывает трудности для проверки эффективности разработанного алгоритма на основе определенных в 2.1 паттернов схожести математических выражений. Ниже, в подразделе 4.1 будут приведены свои корпуса, а в подразделах 4.2 и 4.3 будет рассмотрена эффективность разработанного алгоритма на их основе.

4.1. Составление корпусов для оценки эффективности алгоритма

За источник математических выражений, взятых в основу для составления корпусов, был взят образец 13-го задания ЕГЭ профильного уровня по математике, содержащий тригонометрические уравнения. Набор из 30 заданий из этого образца продемонтрирован в табл. 4.1.

Таблица 4.1. База уравнений

№	Выражение	№	Выражение
1	$\sqrt{2}\sin(\frac{3\pi}{2}-x)\sin x=\cos x$	16	$2\cos(\frac{\pi}{2}+x)=\sqrt{3}\tan x$
2	$\cos(\frac{\pi}{2}+2x)=\sqrt{2}\sin x$	17	$\sin 2x+2\sin^2 x=0$
3	$2\cos(x-\frac{11\pi}{2})\cos x=\sin x$	18	$2\sin(\frac{7\pi}{2}-x)\sin x=\cos x$
4	$2\sin^4 x+3\cos 2x+1=0$	19	$2\sin^2 x-\sqrt{3}\sin 2x=0$
5	$(2\cos x+1)(\sqrt{-\sin x}-1)=0$	20	$\cos 2x-3\cos x+2=0$
6	$(2\sin x-1)(\sqrt{-\cos x}+1)=0$	21	$2\cos^3 x-\cos^2 x+2\cos x-1=0$
7	$4\sin^4 2x+3\cos 4x-1=0$	22	$\cos 2x+3\sin x-2=0$
8	$\cos 2x=\sin(x+\frac{\pi}{2})$	23	$\sin 2x+\sqrt{2}\sin x=2\cos x+\sqrt{2}$
9	$2\sqrt{3}\cos^2(\frac{3\pi}{2}+x)-\sin 2x=0$	24	$3\cos 2x-5\sin x+1=0$
10	$\cos^2 x-\frac{1}{2}\sin 2x+\cos x=\sin x$	25	$\cos 2x-5\sqrt{2}\cos x-5=0$
11	$\cos 2x=1-\cos(\frac{\pi}{2}-x)$	26	$-\sqrt{2}\sin(-\frac{5\pi}{2}+x)\sin x=\cos x$
12	$\sqrt{\cos^2 x-\sin^2 x}(\tan 2x-1)=0$	27	$\frac{2\sin^2 x-\sin x}{2\cos x-\sqrt{3}}=0$

Продолжение табл. 4.1

№	Выражение	№	Выражение
13	$\tan x + \cos(\frac{3\pi}{2} - 2x) = 0$	28	$\frac{2\sin^2 x - \sin x}{2\cos x + \sqrt{3}} = 0$
14	$\cot x + \cos(\frac{\pi}{2} + 2x) = 0$	29	$4\cos^4 x - 4\cos^2 x + 1 = 0$
15	$\frac{1}{2}\sin 2x + sin^2 x - \sin x = \cos x$	30	$4\sin^2 x + 8\sin(\frac{3\pi}{2} + x) + 1 = 0$

Задания были заимствованы с общеобразовательного портала для подготовки абитуриентов [15]. На основе этих уравнений в составе двух экспертов в области подготовки к единому государственному экзамену — автором данной магистерской работы и одним из преподавателей в университете — были составлены:

- корпус, разбивающий рассматриваемые уравнения по классам структурной схожести (табл. 4.2);
- корпус, разбивающий уравнения по классам схожести подвыражений (табл. 4.3).

В ходе составления корпусов, разногласий по отнесению уравнений к тому или иному виду схожести не было. Ниже приведены составленные корпуса.

Таблица 4.2. Корпус, разбивающий уравнения по классам структурной схожести

№	Выражение	Класс структурн. схож.
1	$\sqrt{2}\sin(\frac{3\pi}{2} - x)\sin x = \cos x$	(1)
2	$2\cos(x - \frac{11\pi}{2})\cos x = \sin x$	
3	$2\sin(\frac{7\pi}{2} - x)\sin x = \cos x$	
4	$-\sqrt{2}\sin(-\frac{5\pi}{2} + x)\sin x = \cos x$	
5	$\cos 2x - 3\cos x + 2 = 0$	(2)
6	$\cos 2x + 3\sin x - 2 = 0$	
7	$3\cos 2x - 5\sin x + 1 = 0$	
8	$\cos 2x - 5\sqrt{2}\cos x - 5 = 0$	

Продолжение табл. 4.2

№	Выражение	Класс структурн. схож.
9	$\cos(\frac{\pi}{2}+2x)=\sqrt{2}\sin x$	③
10	$\cos 2x=\sin(x+\frac{\pi}{2})$	
11	$2\cos(\frac{\pi}{2}+x)=\sqrt{3}\tan x$	
12	$2\sin^4 x+3\cos 2x+1=0$	④
13	$4\sin^4 2x+3\cos 4x-1=0$	
14	$4\cos^4 x-4\cos^2 x+1=0$	
15	$(2\cos x+1)(\sqrt{-\sin x}-1)=0$	⑤
16	$(2\sin x-1)(\sqrt{-\cos x}+1)=0$	
17	$\sqrt{\cos^2 x-\sin^2 x}(\tan 2x-1)=0$	
18	$\cos^2 x-\frac{1}{2}\sin 2x+\cos x=\sin x$	⑥
19	$\frac{1}{2}\sin 2x+sin^2x-\sin x=\cos x$	
20	$\tan x+\cos(\frac{3\pi}{2}-2x)=0$	⑦
21	$\cot x+\cos(\frac{\pi}{2}+2x)=0$	
22	$\frac{2\sin^2 x-\sin x}{2\cos x-\sqrt{3}}=0$	⑧
23	$\frac{2\sin^2 x-\sin x}{2\cos x+\sqrt{3}}=0$	

Таблица 4.3. Корпус, разбивающий уравнения по видам схожести подвыражений

№	Выражение	Класс схож. подвыраж.
1	$\cos(\frac{\pi}{2}+2x)=\sqrt{2}\sin x$	①
2	$\cot x+\cos(\frac{\pi}{2}+2x)=0$	
3	$(2\cos x+1)(\sqrt{-\sin x}-1)=0$	②
4	$(2\sin x-1)(\sqrt{-\cos x}+1)=0$	
5	$\sqrt{2}\sin(\frac{3\pi}{2}-x)\sin x=\cos x$	③
6	$2\sin(\frac{7\pi}{2}-x)\sin x=\cos x$	
7	$\frac{2\sin^2 x-\sin x}{2\cos x-\sqrt{3}}=0$	④
8	$\frac{2\sin^2 x-\sin x}{2\cos x+\sqrt{3}}=0$	

Продолжение табл. 4.3

№	Выражение	Класс схож. подвыраж.
9	$2\sin^4 x + 3\cos 2x + 1 = 0$	(5)
10	$4\cos^4 x - 4\cos^2 x + 1 = 0$	
11	$\cos^2 x - \frac{1}{2}\sin 2x + \cos x = \sin x$	
12	$\frac{2\sin^2 x - \sin x}{2\cos x - \sqrt{3}} = 0$	
13	$\frac{2\sin^2 x - \sin x}{2\cos x + \sqrt{3}} = 0$	

Ввиду малого количества выражений в корпусах, алгоритм будет оцениваться лишь с точки зрения точности получаемых результатов. Для этого воспользуемся классической формулой оценки точности:

$$Accuracy = \frac{TP}{TP + FP} \tag{4.1}$$

В формуле 4.1 под TP (true positive) будем понимать количество истинно–положительных первых k найденных уравнений из базы, которые принадлежат к классу запрашиваемого уравнения. FP (false positive) — количество ложно–положительных первых t найденных уравнений из базы, которые принадлежат к классу запрашиваемого уравнения. На k и t вводится ограничение: $k + t = n - 1$, где n — количество уравнений, принадлежащих запрашиваемому уравнению.

Данное ограничение было введено ввиду того, что классы схожести, определенные в корпусах, определяют самые схожие друг с другом уравнения, и не существует точной границы, относящие те или иные уравнения к тому же самому классу.

Ниже приведены примеры подсчета точности поиска алгоритма при поиске структурной схожести и схожести подвыражений на основе формулы 4.1 с учетом описанных выше особенностей.

4.2. Оценка эффективности алгоритма при поиске структурной схожести

Эффективность разработанного алгоритма при поиске структурной схожести оценивалась на одном из составленных корпусов

(табл. 4.2), разделяющим уравнения по классам структурной схожести.

На рис. 4.1 приведен пример подсчета структурной схожести между двумя уравнениями из базы.

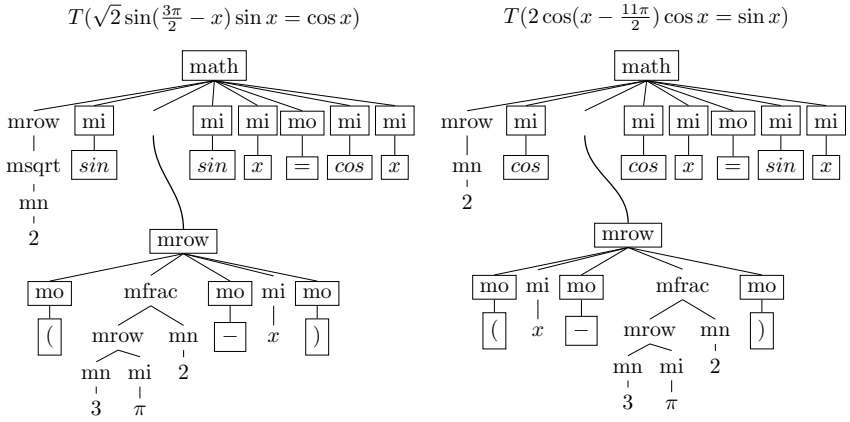

Рисунок 4.1. Подсчет структурной схожести между уравнениями $\sqrt{2}\sin(\frac{3\pi}{2}-x)\sin x=\cos x$ и $2\cos(x-\frac{11\pi}{2})\cos x=\sin x$

На рис. 4.1 мы видим что уравнения имеют 20 общих узлов друг с другом. Уравнение слева имеет 34 узла, а уравнение справа — 33. С применение формулы 2.9 получаем $S_{TO}=\frac{20+20}{34+33}=\frac{40}{67}=0.597$.

В табл. 4.4 приведен результат поиска первых пяти структурно схожих уравнений из базы уравнению $\sqrt{2}\sin(\frac{3\pi}{2}-x)\sin x=\cos x$, принадлежащему к 1-му классу структурно схожих уравнений из корпуса (табл. 4.2).

Поскольку запрашиваемое уравнение принадлежит 1-му классу, а всего уравнений в этом классе 4, то сумма количества истинно–положительных и количества ложно–положительных найденных алгоритмом уравнений не должно превышать $n-1=4-1=3$, поскольку мы ищем самые схожие уравнения. Точность алгоритма при поиске схожих уравнений 1-му уравнению 1-го класса из корпуса будет определяться по формуле 4.1 и равна: $Accuracy_1^1=\frac{TP}{TP+FP}=\frac{2}{2+1}=\frac{2}{3}=0.66$.

Причина того, почему в табл. 4.4 не попало уравнение $-\sqrt{2}\sin(-\frac{5\pi}{2}+x)\sin x=\cos x$ несмотря на то, что оно также было отнесено 1-му классу схожести, будет объяснена ниже в подразделе

Таблица 4.4. Результат поиска первых пяти структурно схожих уравнений из базы уравнению $\sqrt{2}\sin(\frac{3\pi}{2}-x)\sin x=\cos x$, принадлежащему к 1-му классу структурно схожих уравнений

№	Выражение	Кол–во общих узлов	Схож.	Вид
1	$2\sin(\frac{7\pi}{2}-x)\sin x=\cos x$	60 / 67	0.896	(1)
2	$2\cos(x-\frac{11\pi}{2})\cos x=\sin x$	40 / 67	0.597	(1)
3	$2\cos(\frac{\pi}{2}+x)=\sqrt{3}\tan x$	24 / 63	0.381	(3)
4	$3\cos 2x-5\sin x+1=0$	12 / 60	0.200	(2)
5	$\tan x+\cos(\frac{3\pi}{2}-2x)=0$	10 / 67	0.149	(7)

4.4 при анализе результатов.

Вычислим точность поиска структурно схожих уравнений для каждого из уравнений корпуса (табл. 4.5).

Таблица 4.5. Оценка точности структурного поиска

№	Выражение	Точность
1	$\sqrt{2}\sin(\frac{3\pi}{2}-x)\sin x=\cos x$	2/3
2	$2\cos(x-\frac{11\pi}{2})\cos x=\sin x$	2/3
3	$2\sin(\frac{7\pi}{2}-x)\sin x=\cos x$	2/3
4	$-\sqrt{2}\sin(-\frac{5\pi}{2}+x)\sin x=\cos x$	0/3
5	$\cos 2x-3\cos x+2=0$	2/3
6	$\cos 2x+3\sin x-2=0$	2/3
7	$3\cos 2x-5\sin x+1=0$	1/3
8	$\cos 2x-5\sqrt{2}\cos x-5=0$	2/3
9	$\cos(\frac{\pi}{2}+2x)=\sqrt{2}\sin x$	1/2
10	$\cos 2x=\sin(x+\frac{\pi}{2})$	1/2

Продолжение табл. 4.5

№	Выражение	Точность
11	$2\cos(\frac{\pi}{2}+x)=\sqrt{3}\tan x$	0/2
12	$2\sin^4 x+3\cos 2x+1=0$	1/2
13	$4\sin^4 2x+3\cos 4x-1=0$	1/2
14	$4\cos^4 x-4\cos^2 x+1=0$	1/2
15	$(2\cos x+1)(\sqrt{-\sin x}-1)=0$	2/2
16	$(2\sin x-1)(\sqrt{-\cos x}+1)=0$	2/2
17	$\sqrt{\cos^2 x-\sin^2 x}(\tan 2x-1)=0$	2/2
18	$\cos^2 x-\frac{1}{2}\sin 2x+\cos x=\sin x$	0/1
19	$\frac{1}{2}\sin 2x+sin^2x-\sin x=\cos x$	0/1
20	$\tan x+\cos(\frac{3\pi}{2}-2x)=0$	1/1
21	$\cot x+\cos(\frac{\pi}{2}+2x)=0$	1/1
22	$\frac{2\sin^2 x-\sin x}{2\cos x-\sqrt{3}}=0$	1/1
23	$\frac{2\sin^2 x-\sin x}{2\cos x+\sqrt{3}}=0$	1/1

По результатам табл. 4.5, точность нахождения структурно схожих уравнений из базы уравнениям из корпуса равна средней точности определения схожих уравнений из базы каждому уравнению из корпуса и равна $Accuracy=\frac{\frac{2}{3}+\frac{2}{3}+\frac{2}{3}+\frac{0}{3}+\cdots+\frac{1}{1}+\frac{1}{1}}{23}=\frac{13.85}{23}=0.6$.

Причина низкой точности и способ ее увеличения при поиске структурной схожести данным алгоритмом будет объяснена при анализе результатов в параграфе 4.4.1.

4.3. Оценка эффективности алгоритма при поиске схожести подвыражений

Эффективность разработанного алгоритма при поиске схожести подвыражений оценивалась на одном из составленных корпусов

(табл. 4.3), разделяющим уравнения по классам схожести подвыражений.

На рис. 4.2 приведен пример подсчета схожести подвыражений между двумя уравнениями из базы.

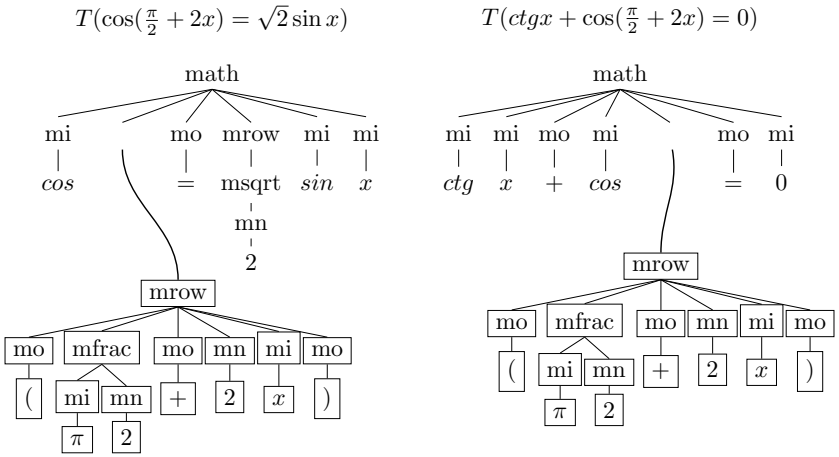

Рисунок 4.2. Подсчет схожести подвыражений между уравнениями $\cos(\frac{\pi}{2}+2x)=\sqrt{2}\sin x$ и $\operatorname{ctg} x+\cos(\frac{\pi}{2}+2x)=0$

На рис. 4.2 мы видим, что уравнения имеют 16 общих узлов друг с другом, образующих общее поддерево. Уравнение слева имеет 29 узлов, а уравнение справа — 30. С применение формулы 2.9 получаем $S_{TO}=\frac{16+16}{29+30}=\frac{32}{59}=0.542$.

В табл. 4.6 приведен результат поиска первых пяти схожих на уровне подвыражений уравнений из базы уравнению $\cos(\frac{\pi}{2}+2x)=\sqrt{2}\sin x$, принадлежащему к 1-му классу схожих на уровне подвыражений уравнений из корпуса (табл. 4.3).

Поскольку запрашиваемое уравнение принадлежит 5-му классу, а всего уравнений в этом классе 5, то сумма количества истинно–положительных и количества ложно–положительных найденных алгоритмом уравнений не должно превышать $n-1=5-1=4$, поскольку мы ищем самые схожие уравнения. Точность алгоритма при поиске схожих уравнений 1-му уравнению 5-го класса из корпуса будет определяться по формуле 4.1 и равна: $Accuracy_1^5=\frac{TP}{TP+FP}=\frac{3}{3+1}=\frac{3}{4}=0.75$.

Причина того, почему уравнение $4\sin^4 2x+3\cos 4x-1=0$ попа-

Таблица 4.6. Результат поиска первых пяти схожих на уровне подвыражений уравнений из базы уравнению $2\sin^4 x + 3\cos 2x + 1 = 0$, принадлежащему к 5-му классу

№	Выражение	Кол–во общих узлов	Схож.	Класс схож. ур-я
1	$4\cos^4 x - 4\cos^2 x + 1 = 0$	10/59	0.169	(5)
2	$4\sin^4 2x + 3\cos 4x - 1 = 0$	10/60	0.167	
3	$\cos^2 x - \frac{1}{2}\sin 2x + \cos x = \sin x$	10/63	0.159	(5)
4	$\frac{2\sin^2 x - \sin x}{2\cos x - \sqrt{3}} = 0$	10/64	0.156	(5)
5	$\frac{2\sin^2 x - \sin x}{2\cos x + \sqrt{3}} = 0$	10/64	0.156	(5)

ло в (табл. 4.3) и повлияло на точность результатов, будет изложена ниже в параграфе 4.4.2.

Подсчитаем точность поиска схожих уравнений на уровне подвыражений для каждого из уравнений корпуса (табл. 4.7).

Таблица 4.7. Оценка точности схожести подвыражений

№	Выражение	Точность
1	$\cos(\frac{\pi}{2} + 2x) = \sqrt{2}\sin x$	1/1
2	$\cot x + \cos(\frac{\pi}{2} + 2x) = 0$	1/1
3	$(2\cos x + 1)(\sqrt{-\sin x} - 1) = 0$	1/1
4	$(2\sin x - 1)(\sqrt{-\cos x} + 1) = 0$	1/1
5	$\sqrt{2}\sin(\frac{3\pi}{2} - x)\sin x = \cos x$	1/1
6	$\sin(\frac{7\pi}{2} - x)\sin x = \cos x$	1/1
7	$\frac{2\sin^2 x - \sin x}{2\cos x - \sqrt{3}} = 0$	1/1
8	$\frac{2\sin^2 x - \sin x}{2\cos x + \sqrt{3}} = 0$	1/1

Продолжение табл. 4.7

№	Выражение	Точность
9	$2\sin^4 x + 3\cos 2x + 1 = 0$	4/5
10	$4\cos^4 x - 4\cos^2 x + 1 = 0$	4/5
11	$\cos^2 x - \frac{1}{2}\sin 2x + \cos x = \sin x$	4/5
12	$\frac{2\sin^2 x - \sin x}{2\cos x - \sqrt{3}} = 0$	5/5
13	$\frac{2\sin^2 x - \sin x}{2\cos x + \sqrt{3}} = 0$	5/5

По результатам табл. 4.7, точность нахождения схожих на уровне подвыражений уравнений из базы уравнениям из корпуса равна средней точности определения схожих уравнений из базы каждому уравнению из корпуса и равна $Accuracy = \frac{\frac{1}{1}+\frac{1}{1}+\cdots+\frac{4}{5}+\frac{5}{5}+\frac{5}{5}}{13} = \frac{12.4}{13} = 0.95$.

4.4. Анализ результатов оценки эффективности алгоритма

4.4.1. Анализ результатов по оценке алгоритма при поиске структурной схожести

В подразделе 4.2 была определена точность алгоритма при поиске структурной схожести и составила 0.6%. Такой низкой точности послужила особенность в синтаксической конструкции выражений, представленных в формате MathML, по которым производился поиск структурной схожести.

Рассмотрим уравнения $\sqrt{2}\sin(\frac{3\pi}{2} - x)\sin x = \cos x$ и $-\sqrt{2}\sin(-\frac{5\pi}{2} + x)\sin x = \cos x$, принадлежащие к 1-му классу структурной схожести по разработанному корпусу. В результате поиска по разработанному алгоритму, схожести между ними замечено не было (рис. 4.3).

Согласно паттерну структурной схожести выражений 2.1, для того, чтобы выражения были структурно схожи, они должны иметь общие самые верхние узлы с одинаковым названием и одинаковыми правилами перехода к последующим узлам. У самых верхних узлов на

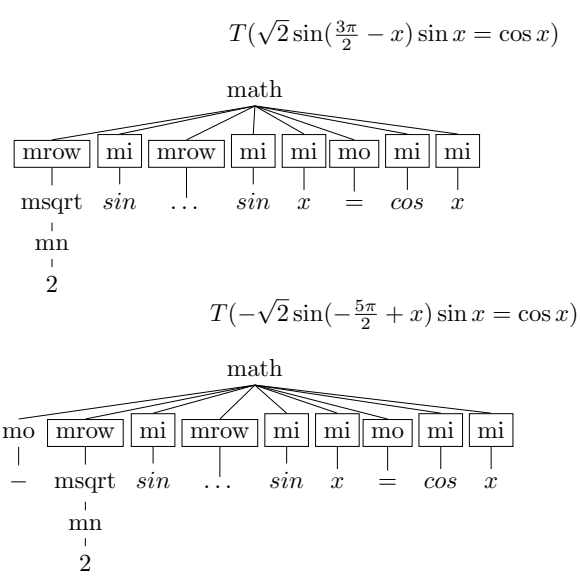

Рисунок 4.3. Подсчет структурной схожести между уравнениями $\sqrt{2}\sin(\frac{3\pi}{2} - x)\sin x = \cos x$ и $-\sqrt{2}\sin(-\frac{5\pi}{2} + x)\sin x = \cos x$

рис. 4.3 разное количество последующих узлов и, как следствие, разные правила перехода. Ввиду этого, несмотря на визуально схожую структуру и отнесению экспертами уравнений $\sqrt{2}\sin(\frac{3\pi}{2} - x)\sin x = \cos x$ и $-\sqrt{2}\sin(-\frac{5\pi}{2} + x)\sin x = \cos x$ к одному классу структурной схожести, в ходе выполнения поиска по разработанному алгоритму их структурная схожесть не определяется.

На рис. 4.4 можно увидеть рассматриваемые уравнения в двух разных видах — изначальном (T_1 и T_2) и имеющим другую структурную конструкцию ($T_1^{'}$ и $T_2^{'}$), но визуально и семантически представляемых одинаково. Тем не менее, их синтаксическая схожесть в изначальном представлении не будет определена, а в модифицированном будет равна 0.44.

С целью выявления синтаксической схожести выражений, имеющих разное количество узлов, начиная от коренных узлов (пример: выражения T_1 и T_2 на рис. 4.4), данные выражения необходимо предварительно преобразовать (пример: выражения $T_1^{'}$ и $T_2^{'}$ на рис. 4.4),

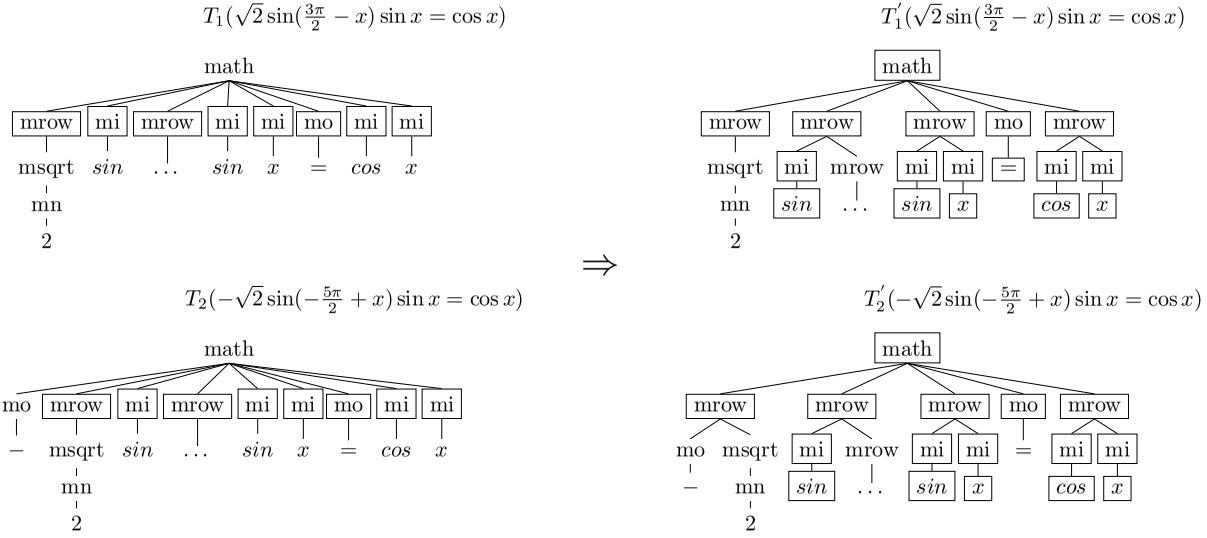

Рисунок 4.4. Пример перехода от одних сравниваемых выражений к другим, имеющим другую синтаксическую структуру, но эквивалентным изначальным с точки зрения визуального представления и семантического смысла с целью поиска их синтаксической схожести

выделяя необходимые конструкции в выражениях в отдельные подвыражения. Для этого предлагается один из следующих подходов:

1. выбрать другой конвертер для конвертации выражений из LaTeX в формат MathML, который бы выделял необходимые конструкции в отдельные подвыражения;

2. при конвертации выражений из формата LaTeX в формат MathML используемым конвертером [14] нарочно добавлять фигурные скобки, ограничивающие конструкции, которые должны быть выделены в подвыражения: этот способ и был использован на рис. 4.4;

   - $ \sqrt{2} \sin (\frac{3\pi}{2} - x) \sin x = \cos x $ ⇒
     $ {\sqrt{2}} {\sin (\frac{3\pi}{2} - x)} {\sin x} = {\cos x} $

   - $ -\sqrt{2} \sin (\frac{5\pi}{2} + x) \sin x = \cos x $ ⇒
     $ {-\sqrt{2}} {\sin (\frac{5\pi}{2} + x)} {\sin x} = {\cos x} $

3. производить анализ выражения, уже представленного в формате MathML, и по определенному алгоритму выделять необходимые конструкции в подвыражения: данный вариант является более перспективным в силу того, что уже существующие выражения, представленные в MathML, не нужно будет пересоздавать заново.

После анализа особенности представления выражений в формате MathML были внесены поправки во все выражения из базы перед конвертацией их из LaTeX в формат MathML с помощью 2-ого подхода. После этого структурный поиск при повторном тестировании на том же корпусе был абсолютно точен и составил 100%.

4.4.2. Анализ результатов по оценке алгоритма при поиске схожести подвыражений

На ухудшение точности поиска подвыражений в подразделе 4.3 повлияло уравнение, найденное вторым в табл. 4.3, которое не относилось к тому же классу схожести подвыражений запрашиваемого

уравнения. Такой результат был получен ввиду того, что одно и то же уравнение может содержать несколько подвыражений, которые могут в свою очередь быть общими с подвыражениями разных уравнений. Рассмотрим запрашиваемое уравнение подробнее:

- $2\sin^4 x + 3\boxed{\cos 2x} + 1 = 0$

Серым цветом обозначена область, которая повлияла на мнение экспертов, которые составляли корпус, чтобы отнести уравнение к определенному классу схожести подвыражений. В тоже время не было учтено то, что у того же самого уравнения есть другое подвыражение, выделенное в рамку, которое также может отнести уравнение и к другому классу схожести. При поиске и произошла подобная ситуация:

- $2\sin^4 x + 3\boxed{\cos 2x} + 1 = 0$
- $4\sin^4 2x + 3\boxed{\cos 4x} - 1 = 0$

Второе уравнение схоже с первым в подвыражении, выделенным в рамку, а не в подвыражении, выделенным серым цветом. В силу этой особенности точность поиска схожести подвыражений алгоритма на составленном корпусе составила 0.95%.

4.5. Общие выводы относительно полученных результатов и рекомендации по применению алгоритма

Оценка точности поиска разработанным алгоритмом содержала ряд трудностей, изложенных в параграфах 4.4.1 и 4.4.2. Тем не менее, после внесения ряда изменений с учетом этих особенностей, алгоритм показал высокую точность поиска структурной схожести и схожести подвыражений между заданным выражением и базой выражений (табл. 4.1) на составленных корпусах (табл. 4.2 и табл. 4.3).

Полученные результаты относительно эффективности разработанного алгоритма можно считать успешными, и позволяют рекомендовать данный алгоритм к встраиванию в поисковые системы, нацеленные на работу с математической нотацией в формате MathML с разметкой Presentation Markup.

Преимущества данного алгоритма перед существующими [1], [2] по поиску синтаксической схожести между математическими выражениями, представленными в формате MathML:

- разделяет синтаксическую схожесть на подвиды (структурная схожесть и схожесть подвыражений) и позволяет ее находить;
- позволяет получить дополнительную информацию о схожих выражениях, такую как: кол-во общих узлов, коренной узел общей области, является ли эта общая область поддеревом;
- производит предварительную индексацию корпуса деревьев, по которым будет производиться поиск с целью ускорения поиска, которая также присутствует только в одной [1] из двух исследованных реализаций.

Можно привести следующие направления для дальнейшего развития алгоритма:

- введение весов на узлы деревьев выражений для выделения значимости тех или иных областей при поиске схожести подвыражений;
- расширение области его применения до поиска схожести между математическими выражениями, представленными в формате MathML с разметкой Content Markup;
- визуальное представление схожести выражений при отображении выражений.

ЗАКЛЮЧЕНИЕ

В данной работе определены паттены синтаксической схожести между выражениями, представленными в формате MathML. Был модифицирован существующий алгоритм нахождения схожести между упорядоченными ориентированными деревьями для адаптации его к поиску синтаксически схожих математических выражений. Было разработано программное обеспечение, реализующее поиск схожих математических выражений на основе модифицированного алгоритма.

Для проверки эффективности разработанного алгоритма, тестирование полученных результатов проводилось на корпусах, составленных, из уравнений, взятых с сайта для подготовки абитуриентов к сдаче единого государственного экзамена. В ходе тестирования обнаружилилась особенность в инвариантности древовидного представления математических выражений после их конвертации из формата LaTeX в формат MathML. В силу этого, при определенной структуре тестируемых выражений, алгоритм демострирует средние результаты по поиску, в остальных случаях алгоритм с высокой степенью точности определяет схожие выражения. Анализ результатов тестирования позволил выявить направления для дальнейшего совершенствования алгоритма.

К отличительным особенностям данного алгоритма можно отнести возможность задавать тип схожести (структурная схожесть или схожесть подвыражений) при поиске синтаксической схожести между выражениями.

Разработанный алгоритм может применяться в системах автоматизации сортировки схожих математических выражений для пополнения их к уже существующей базе примеров, системах по составлению тестовых заданий или задачников по математике, поиске схожих математических выражений по шаблону.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Yokoi Keisuke, Aizawa Akiko. An approach to similarity search for mathematical expressions using MathML // Towards a Digital Mathematics Library. Grand Bend, Ontario, Canada, July 8-9th, 2009. — 2009. — P. 27–35.
2. Sain Kunal, Dasgupta Abhishek, Garain Utpal. EMERS: a tree matching–based performance evaluation of mathematical expression recognition systems // International Journal on Document Analysis and Recognition (IJDAR). — 2011. — Vol. 14, no. 1. — P. 75–85.
3. World Wide Web Consortium (W3C) and International Digital Publishing Forum. — http://www.w3.org/TR/MathML2/overview.html.
4. Zhang Kaizhong, Shasha Dennis. Simple fast algorithms for the editing distance between trees and related problems // SIAM journal on computing. — 1989. — Vol. 18, no. 6. — P. 1245–1262.
5. Klein Philip N. Computing the edit-distance between unrooted ordered trees. — 1998. — P. 91–102.
6. An optimal decomposition algorithm for tree edit distance / Erik D Demaine, Shay Mozes, Benjamin Rossman, Oren Weimann // ACM Transactions on Algorithms (TALG). — 2009. — Vol. 6, no. 1. — P. 2.
7. Pawlik Mateusz, Augsten Nikolaus. RTED: a robust algorithm for the tree edit distance // Proceedings of the VLDB Endowment. — 2011. — Vol. 5, no. 4. — P. 334–345.
8. Efficient sentence retrieval based on syntactic structure / Ichikawa Hiroshi, Hakoda Keita, Hashimoto Taiichi, Tokunaga Takenobu. — 2006. — P. 399–406.
9. Collins Michael, Duffy Nigel, Park Florham. Parsing with a single neuron: Convolution kernels for natural language problems. — 2001.
10. Bod Rens. Beyond grammar // An Experienced-Based Theory of Language. CSLI Lecture Notes. — 1998. — Vol. 88.
11. The Wolfram Functions Site, Wolfram Research Inc. — http://functions.wolfram.com.
12. Garain Utpal. Automatic recognition of printed and handwritten mathematical expressions. — 2005.
13. Kamali Shahab, Tompa Frank Wm. Improving mathematics re-

trieval // Towards a Digital Mathematics Library. Grand Bend, Ontario, Canada, July 8-9th, 2009. — 2009. — P. 37–48.
14. Convert LaTeX to Presentation MathML. — URL: https://www.mathtowebonline.com.
15. РЕШУ ЕГЭ. Образовательный портал для подготовки к экзаменам. — URL: https://math-ege.sdamgia.ru.

ПРИЛОЖЕНИЕ А
ТЕКСТ ПРОГРАММЫ

Листинг А.1. Исходный код класса SearchEngine

```
1  package searchengine;
2
3  import additional.Pair;
4  import entities.IdxNode;
5  import entities.MathMLTree;
6  import entities.ProductionRule;
7  import entities.PRwithIdxNodes;
8  import entities.Sto;
9  import java.util.ArrayList;
10 import java.util.Collections;
11 import java.util.Comparator;
12 import java.util.Enumeration;
13 import java.util.HashMap;
14 import java.util.Iterator;
15 import java.util.LinkedHashMap;
16 import java.util.List;
17 import java.util.Map;
18
19
20
21 public class SearchEngine {
22
23     private List<PRwithIdxNodes<String>> corpusIdxTable;
24     private List<PRwithIdxNodes<String>> queryIdxTable;
25     private List<MathMLTree> corpusTrees;
26     private MathMLTree queryTree;
27
28     private ArrayList<Sto> StoList;
29
30
31     public static enum StoOrder {
32         SIMILARITY,
33         TREEID;
34     }
35
36     public ArrayList<Sto> getStoList(StoOrder order) {
37
38         switch(order) {
39             case SIMILARITY:
40                 Collections.sort(StoList, new Comparator<Sto>() {
41                     @Override
42                     public int compare(Sto o1, Sto o2) {
43                         if (o1.simValue() < o2.simValue()) return -1;
44                         if (o1.simValue() > o2.simValue()) return 1;
45                         return 0;
46                     }
47                 });
48                 break;
49             case TREEID:
```

```
50                  Collections.sort(StoList, new Comparator<Sto>() {
51                      @Override
52                      public int compare(Sto o1, Sto o2) {
53                          return o1.tr2.getTreeID() - o2.tr2.getTreeID();
54                      }
55                  });
56          }
57          Collections.reverse(StoList);
58
59
60          return this.StoList;
61      }
62
63
64      public void printStoList(StoOrder order) {
65          if(StoList.isEmpty())
66              System.out.println("StoList is empty!");
67
68          switch(order) {
69              case SIMILARITY:
70                  Collections.sort(StoList, new Comparator<Sto>() {
71                      @Override
72                      public int compare(Sto o1, Sto o2) {
73                          if (o1.simValue() < o2.simValue()) return -1;
74                          if (o1.simValue() > o2.simValue()) return 1;
75                          return 0;
76                      }
77                  });
78                  break;
79              case TREEID:
80                  Collections.sort(StoList, new Comparator<Sto>() {
81                      @Override
82                      public int compare(Sto o1, Sto o2) {
83                          return o1.tr2.getTreeID() - o2.tr2.getTreeID();
84                      }
85                  });
86                  //do nothing
87          }
88
89
90
91
92          for(Sto s : StoList)
93              s.print(Sto.ViewStyle.OVERLAYNODES);
94      }
95
96
97
98
99      public SearchEngine(MathMLTree quTree) {
100         this.corpusIdxTable = new ArrayList<>();
101         this.queryIdxTable = new ArrayList<>();
102         this.corpusTrees = new ArrayList<>();
103         this.queryTree = null;
104         this.StoList = new ArrayList<>();
105
106         setQueryTree(quTree);
107     }
```

```
108
109     public SearchEngine() {
110         this.corpusIdxTable = new ArrayList<>();
111         this.queryIdxTable = new ArrayList<>();
112         this.corpusTrees = new ArrayList<>();
113         this.queryTree = null;
114         this.StoList = new ArrayList<>();
115     }
116
117
118
119
120     public void setQueryTree(MathMLTree quTree) {
121         quTree.setTreeIDSameForAllTree(0); // changed!!!
122         this.queryTree = quTree;
123         addTreeToIndexTable(quTree, queryIdxTable);
124     }
125
126     // должны добавляться в порядке возврастания индекса дерева!
127     public void addCorpusTree(MathMLTree corpTree) {
128
129         if(corpTree.getTreeID() != 0)
130             corpTree.setTreeIDSameForAllTree(corpusTrees.size() + 1);
                    // changed!!!
131
132         corpusTrees.add(corpTree);
133         this.addTreeToIndexTable(corpTree, corpusIdxTable);
134     }
135
136     public MathMLTree getTreeFromCorpusTreesByIndexNode(IdxNode<String>
          idxNode, List<MathMLTree> cTrs) {
137         for(MathMLTree tr : cTrs)
138             if(tr.getTreeID() == idxNode.getTreeID())
139                 return tr;
140
141         return null;
142     }
143
144
145     private void addTreeToIndexTable(MathMLTree tree, List<
          PRwithIdxNodes<String>> idxTbl) {
146         /*
147         Алгоритм:
148             1. Проходимся по всем узлам в ширину
149             2. Prod Rule текущего узла в таблице:
150                 если есть  ->
151                     Corp Node текущего узла в списке Corp Node этого PR
152                     если есть  -> не добавляем
153                     если нету  -> добавляем
154                 если нету  ->
155                     а) добавляем PR
156                     б) добавляем к PR текущий узел
157         */
158
159         MathMLTree node;              // текущий узел
160         ProductionRule<String> PR;   // текущий PR узла
161         IdxNode<String> CN;         // текущий CN узла
```

```
162         boolean hasPR;              // нашелся ли в таблице индексов PR
                узла
163
164
165         // проходимся по всем элементам дерева в ширину
166         for(Enumeration e = tree.breadthFirstEnumeration(); e.
                hasMoreElements();) {
167
168             node = (MathMLTree)e.nextElement();
169             PR = new ProductionRule<>(node);
170             CN = new IdxNode<>(node);
171             hasPR = false;
172
173             // проходимся по Index Table
174             for (PRwithIdxNodes<String> corpIdx : idxTbl)
175                 if(corpIdx.getPR().equals(PR)) {
176                     hasPR = true;
177                     if(containsIdenticalInCorpusTreeNodes(corpIdx.
                            getIdxNodes(), CN))
178                         break;
179                     else {
180                         corpIdx.getIdxNodes().add(CN);
181                         break;
182                     }
183                 }
184
185             // если нету в базе такого PR
186             if(!hasPR) {
187                 PRwithIdxNodes<String> newCorpIdx = new PRwithIdxNodes
                        <>(PR);
188                 newCorpIdx.add_topCorpusTreeNode(CN);
189                 idxTbl.add(newCorpIdx);
190             }
191         }
192
193     }
194
195     // получаем списочек Cto
196     public Map<Pair<IdxNode<String>,IdxNode<String>>, Pair<Integer,
            Integer>>
197         calculateCtoList(MathMLTree qTr, List<PRwithIdxNodes<String>>
                qIdxTable,
198                 List<MathMLTree> cTrs, List<PRwithIdxNodes<String>>
                        corpIdxTable) {
199
200         Map<Pair<IdxNode<String>,IdxNode<String>>, Pair<Integer,
                Integer>>
201             Cto = new LinkedHashMap<>();
202
203         for(PRwithIdxNodes<String> qPRwNodes : qIdxTable)
204             for(PRwithIdxNodes<String> cPRwNodes : corpIdxTable)
205                 if(qPRwNodes.getPR().equals(cPRwNodes.getPR()))
206                     for(IdxNode<String> qIdxNode : qPRwNodes.
                            getIdxNodes())
207                         for(IdxNode<String> cIdxNode : cPRwNodes.
                                getIdxNodes())
208                             if(qIdxNode.equals(IdxNode.Content.
                                    label_leavesCount, cIdxNode)) {
```

```
209
210                                 MathMLTree cTr =
                                        getTreeFromCorpusTreesByIndexNode(
                                        cIdxNode, cTrs);
211                                 Pair topNM = topIndexNode(qIdxNode, qTr
                                        , cIdxNode, cTr);
212
213
214 //                                  printTopColumn(qPRwNodes.getPR().
        getStringView(),
215 //                                      qIdxNode.getStringView(
        IdxNode.Content.label_repeatID, true),
216 //                                      cIdxNode.getStringView(
        IdxNode.Content.label_repeatID, true),
217 //                                      ((IdxNode)topNM.n).
        getStringView(IdxNode.Content.label_repeatID, true),
218 //                                      ((IdxNode)topNM.m).
        getStringView(IdxNode.Content.label_repeatID, true),
219 //                                      (qIdxNode.getLeavesCount() ==
         0));
220
221                                 // сколько раз встретился корень дерева
222                                 int leavesCover = 0;
223                                 // сколько раз встретился узел
224                                 int overlap = 0;
225                                 boolean leafCover = (qIdxNode.
                                        getLeavesCount() == 0) &&
226                                             (((IdxNode)topNM.n).
                                                 getLeavesCount() != 0);
227
228
229                                 if(Cto.containsKey(topNM)) {
230                                     overlap = Cto.get(topNM).n + 1;
231                                     if(leafCover)
232                                         leavesCover = Cto.get(topNM).m
                                                + 1;
233                                     else
234                                         leavesCover = Cto.get(topNM).m;
235                                 }
236                                 else {
237                                    overlap = 1;
238                                    if(leafCover)
239                                        leavesCover = 1;
240                                    else
241                                        leavesCover = 0;
242                                 }
243
244                                 // без проверки того, сколько листьев н
                                        алажено
245                                 Cto.put(topNM, new Pair(overlap,
                                        leavesCover));
246                             }
247         return Cto;
248     }
249
250
251     // RECURSIVE METHOD!
```

```
252     public Map<Pair<IdxNode<String>, IdxNode<String>>, Pair<Integer,
            Integer>>
253         Cto_to_CtoSubtrList(MathMLTree tr0, MathMLTree tr1,
254             IdxNode idxNode1, IdxNode idxNode2,
255             Map<Pair<IdxNode<String>, IdxNode<String>>, Pair<Integer,
                    Integer>> CtoSubtrCollector) {
256         //System.out.println("RECURSIVE METHOD: Cto_to_CtoSubtrList");
257
258
259         // получаем поддеревья, начинающиеся с узлов idxNode1 и
                idxNode2
260         MathMLTree t0 = (MathMLTree) tr0.getNodeByBreadthOrderIndex(
                idxNode1.getBreadthPosInTree());
261         MathMLTree t1 = (MathMLTree) tr1.getNodeByBreadthOrderIndex(
                idxNode2.getBreadthPosInTree());
262
263         // копируем поддеревья
264         MathMLTree t0_subtr = MathMLTree.fromMathMLTree(t0);
265         MathMLTree t1_subtr = MathMLTree.fromMathMLTree(t1);
266
267         // изменяем корни поддеревьев, чтобы Cto в них еще раз не было
268         t0_subtr.setLabel("XYZ");
269         t1_subtr.setLabel("RTY");
270
271         // делаем таблицу индексов для поддеревьев
272         List<PRwithIdxNodes<String>> t0SubtrIdxTable = new ArrayList
                <>();
273         List<PRwithIdxNodes<String>> t1SubtrIdxTable = new ArrayList
                <>();
274         addTreeToIndexTable(t0_subtr, t0SubtrIdxTable);
275         addTreeToIndexTable(t1_subtr, t1SubtrIdxTable);
276
277         // .. для метода
278         List<MathMLTree> t1_subtr_list = new ArrayList<>();
279         t1_subtr_list.add(t1_subtr);
280
281
282         // получаем список новых Cto
283         Map<Pair<IdxNode<String>, IdxNode<String>>, Pair<Integer,
                Integer>>
284                 CtoList = calculateCtoList(t0_subtr,
                        t0SubtrIdxTable, t1_subtr_list,
                        t1SubtrIdxTable);
285
286
287         for(Iterator<Map.Entry<Pair<IdxNode<String>, IdxNode<String>>,
                Pair<Integer, Integer>>>
288             it = CtoList.entrySet().iterator(); it.hasNext(); ) {
289             Map.Entry<Pair<IdxNode<String>, IdxNode<String>>, Pair<
                    Integer, Integer>>
290                 Cto = it.next();
291
292             if(Cto.getValue().m != Cto.getKey().n.getLeavesCount()) {
293                 Cto_to_CtoSubtrList(t0_subtr, t1_subtr, Cto.getKey().n,
                        Cto.getKey().m, CtoSubtrCollector);
294                 it.remove();
295             } else {
296                 CtoSubtrCollector.put(Cto.getKey(), Cto.getValue());
```

```
297             }
298         }
299
300
301         return CtoList;
302     }
303
304
305         // схожесть подвыражений
306     public void calculateStructuralSimilarity() {
307
308
309         Map<Pair<IdxNode<String>,IdxNode<String>>, Pair<Integer,
                Integer>> CtoList;
310         // Cto (с теми узлами, которые подходят по покрытию листьев)
311         Map<Pair<IdxNode<String>,IdxNode<String>>, Integer>
                Cto_WithooutLeaveInfo = new HashMap<>();
312
313         // key      = pair (number of 1-st tree,
314         //                  number of 2-st tree)
315         // value    = pair (overlap,
316         //                  pair (idxNode of 1-st tree,
317         //                        idxNode of 2-nd tree)
318         Map<Pair<Integer, Integer> ,
319             Pair <Integer, Pair<IdxNode<String>,IdxNode<String>>>>
                    StoRoot = new HashMap<>();
320
321
322         // Cto - классическое Cto по алгоритму из статьи
323         CtoList = calculateCtoList(queryTree, queryIdxTable,
                corpusTrees, corpusIdxTable);
324
325
326         // Cto_WithooutLeaveInfo <- Cto
327         for(Map.Entry<Pair<IdxNode<String>,IdxNode<String>>, Pair<
                Integer, Integer>> entry
328                 : CtoList.entrySet())
329                 Cto_WithooutLeaveInfo.put(entry.getKey(), entry.
                        getValue().n);
330
331
332         // Sto_Subtr - наибольшее по Cto_Subtr
333         for(Map.Entry<Pair<IdxNode<String>,IdxNode<String>>,Integer>
                entry
334                 : Cto_WithooutLeaveInfo.entrySet()) {
335
336             int NtrID = entry.getKey().n.getTreeID();
337             int MtrID = entry.getKey().m.getTreeID();
338             Pair pairTrID = new Pair(NtrID, MtrID);
339
340             if(entry.getKey().m.getLabel().equals("math"))
341                 StoRoot.put(pairTrID, new Pair(entry.getValue(), entry.
                        getKey()));
342         }
343
344
345         // запихиваем результаты в отдельный объект для этого
```

```
346         for(Map.Entry<Pair<Integer, Integer>, Pair<Integer, Pair<
                IdxNode<String>, IdxNode<String>>>> entry
347                 : StoRoot.entrySet()) {
348
349             Sto sim = new Sto(
350                     entities.Sto.Type.STRUCTURAL,
351                     queryTree,
352                     entry.getValue().m.n,
353                     getTreeFromCorpusTreesByIndexNode(entry.getValue().
                            m.m, corpusTrees),
354                     entry.getValue().m.m,
355                     entry.getValue().n);
356
357             this.StoList.add(sim);
358         }
359
360         addALLMissedStoToStoList(Sto.Type.STRUCTURAL);
361     }
362
363
364
365     private void addALLMissedStoToStoList(Sto.Type type) {
366
367
368         for(MathMLTree tr : corpusTrees) {
369             boolean hasSto = false;
370             for(Sto Sto : StoList)
371                 if(Sto.tr2.getTreeID() == tr.getTreeID())
372                    hasSto = true;
373
374             if(!hasSto)
375                 StoList.add(new Sto(type, queryTree, tr));
376         }
377     }
378
379
380
381     // схожесть подвыражений
382     public void calculateSubExprSimilarity() {
383
384
385         Map<Pair<IdxNode<String>,IdxNode<String>>, Pair<Integer,
                Integer>> CtoList;
386         // Cto (с теми узлами, которые подходят по покрытию листьев)
387         Map<Pair<IdxNode<String>,IdxNode<String>>, Integer> Cto_Subtr =
                new HashMap<>();
388
389         // key      = pair (number of 1-st tree,
390         //                  number of 2-st tree)
391         // value    = pair (overlap,
392         //                  pair (idxNode of 1-st tree,
393         //                        idxNode of 2-nd tree)
394         Map<Pair<Integer, Integer> ,
395             Pair <Integer, Pair<IdxNode<String>,IdxNode<String>>>>
                    Sto_Subtr = new HashMap<>();
396
397         Map<Pair<IdxNode<String>,IdxNode<String>>, Pair<Integer,
                Integer>> Cto_subtr_collector = new LinkedHashMap<>();
```

```
398
399
400             // Cto - классическое Cto по алгоритму из статьи
401             CtoList = calculateCtoList(queryTree, queryIdxTable,
                    corpusTrees, corpusIdxTable);
402
403
404             //  РЕКУРСИВНО заменяем Cto, которые не покрывают поддерево
405             for(Map.Entry<Pair<IdxNode<String>,IdxNode<String>>, Pair<
                    Integer, Integer>> Cto
406                     : CtoList.entrySet()) {
407
408                 if(Cto.getValue().m != Cto.getKey().n.getLeavesCount()) {
409                     MathMLTree qTr = queryTree;
410                     MathMLTree cTr = getTreeFromCorpusTreesByIndexNode(Cto.
                            getKey().m, corpusTrees);
411                     Cto_to_CtoSubtrList(queryTree, cTr, Cto.getKey().n, Cto
                            .getKey().m, Cto_subtr_collector);
412                 } else {
413                     Cto_subtr_collector.put(Cto.getKey(), Cto.getValue());
414                 }
415             }
416
417
418             // Cto_Subtr - Cto, которые подходят по покрытию (тут должны по
                    ходить все)
419             for(Map.Entry<Pair<IdxNode<String>,IdxNode<String>>, Pair<
                    Integer, Integer>> entry
420                     : Cto_subtr_collector.entrySet())
421                 if(entry.getValue().m == entry.getKey().n.getLeavesCount())
422                     Cto_Subtr.put(entry.getKey(), entry.getValue().n);
423
424
425             // Sto_Subtr - наибольшее по Cto_Subtr
426             for(Map.Entry<Pair<IdxNode<String>,IdxNode<String>>,Integer>
                    entry
427                     : Cto_Subtr.entrySet()) {
428
429                 int NtrID = entry.getKey().n.getTreeID();
430                 int MtrID = entry.getKey().m.getTreeID();
431                 Pair pairTrID = new Pair(NtrID, MtrID);
432                 int overlap = entry.getValue();
433
434                 if(Sto_Subtr.containsKey(pairTrID)) {
435                     if(Sto_Subtr.get(pairTrID).n < overlap)
436                         Sto_Subtr.put(pairTrID, new Pair(overlap, entry.
                                getKey()));
437                 } else {
438                     Sto_Subtr.put(pairTrID, new Pair(overlap, entry.getKey
                            ()));
439                 }
440             }
441
442
443             // запихиваем результаты в отдельный объект для этого
444             for(Map.Entry<Pair<Integer, Integer>, Pair<Integer, Pair<
                    IdxNode<String>, IdxNode<String>>>> entry
445                     : Sto_Subtr.entrySet()) {
```

```
446
447             Sto sim = new Sto(
448                     entities.Sto.Type.SUBEXPRESSION,
449                     queryTree,
450                     entry.getValue().m.n,
451                     getTreeFromCorpusTreesByIndexNode(entry.getValue().
                            m.m, corpusTrees),
452                     entry.getValue().m.m,
453                     entry.getValue().n);
454
455             this.StoList.add(sim);
456         }
457     }
458
459
460
461
462
463     private void printTopColumn(String PR, String n, String m, String
            topN, String topM, boolean getLC) {
464
465         System.out.print(PR);
466         System.out.print("  : (");
467         System.out.print(n);
468         System.out.print(",");
469         System.out.print(m);
470         System.out.print(")");
471
472         System.out.print("  top(");
473         System.out.print(topN);
474         System.out.print(",");
475         System.out.print(topM);
476         System.out.print(")");
477         System.out.println(" (" + getLC + ")");
478     }
479
480
481
482
483     private void printCto_SubtrList(Map<Pair<IdxNode<String>,IdxNode<
            String>>, Integer> Cto) {
484
485         System.out.println("CtoList_Subtr: ");
486         for(Map.Entry<Pair<IdxNode<String>,IdxNode<String>>, Integer>
                entry  : Cto.entrySet()) {
487             System.out.print("(");
488             System.out.print(entry.getKey().n.getStringView(IdxNode.
                    Content.
                    label_treeID_repeatID_leavesCount_breadthPosInTree,
                    true));
489             System.out.print(",");
490             System.out.print(entry.getKey().m.getStringView(IdxNode.
                    Content.
                    label_treeID_repeatID_leavesCount_breadthPosInTree,
                    true));
491             System.out.print(") ");
492             System.out.println("overlap: " + entry.getValue());
493         }
```

```
494     }
495
496
497     private void printCtoList(Map<Pair<IdxNode<String>,IdxNode<String
            >>,
498             Pair<Integer, Integer>> preCto) {
499
500             System.out.println("CtoList: ");
501
502             for(Map.Entry<Pair<IdxNode<String>,IdxNode<String>>, Pair<
                    Integer, Integer>> entry
503                     : preCto.entrySet()) {
504             System.out.print("(");
505             System.out.print(entry.getKey().n.getStringView(IdxNode.
                    Content.
                    label_treeID_repeatID_leavesCount_breadthPosInTree,
                    true));
506             System.out.print(",");
507             System.out.print(entry.getKey().m.getStringView(IdxNode.
                    Content.
                    label_treeID_repeatID_leavesCount_breadthPosInTree,
                    true));
508             System.out.print(") ");
509             System.out.print("overlap: " + entry.getValue().n);
510             System.out.print(", leavesCover: " + entry.getValue().m);
511             System.out.println(" / " + entry.getKey().n.getLeavesCount
                    ());
512         }
513     }
514
515
516
517     // n - узел дерева tr_n
518     // m - узел дерева tr_m
519     public Pair<IdxNode<String>,IdxNode<String>>
520          topIndexNode(IdxNode<String> n, MathMLTree tr_n,
521              IdxNode<String> m, MathMLTree tr_m) {
522
523         if(!n.equals(IdxNode.Content.label, m)) {
524             System.err.println("Имена начальных узлов не совпадают!");
525             return null;
526         }
527
528         MathMLTree N = (MathMLTree) tr_n.getNodeByBreadthOrderIndex(n.
                getBreadthPosInTree());
529         MathMLTree M = (MathMLTree) tr_m.getNodeByBreadthOrderIndex(m.
                getBreadthPosInTree());
530
531
532         if(!N.getLabel().equals(n.getLabel()))
533             System.err.println("Имена index node и узла в дереве на сов
                    падают!");
534         if(!M.getLabel().equals(m.getLabel()))
535             System.err.println("Имена index node и узла в дереве на сов
                    падают!");
536
537
538         if(N.isRoot() || M.isRoot())
```

```
539             return new Pair(n,m);
540
541
542         Pair<MathMLTree, MathMLTree> NM     = new Pair<>(N, M);
543         Pair<MathMLTree, MathMLTree> newNM  = top(N, M);
544
545
546
547         while(!NM.equals(newNM)) {
548             NM = newNM;
549             newNM = top(NM.n, NM.m);
550         }
551
552
553         Pair p = new Pair<>(new IdxNode(NM.n),
554                             new IdxNode(NM.m));
555
556         return p;
557     }
558
559
560     private Pair<MathMLTree, MathMLTree> top(MathMLTree t1, MathMLTree
            t2) {
561
562         // если названия t1 и t2 не совпадают изначально
563         if(!t1.getLabel().equals(t2.getLabel())) {
564             System.err.println("Имена не совпали!");
565             return null;
566         }
567
568         // если они уже руты
569         if(t1.isRoot() || t2.isRoot()) {
570             return new Pair(t1, t2);
571         }
572
573         String pT1 = ((MathMLTree)t1.getParent()).getLabel();
574         String pT2 = ((MathMLTree)t2.getParent()).getLabel();
575
576         // если совпадают имена предыдущего узла и порядок среди братье
                в и систер
577         if(pT1.equals(pT2))
578             if(t1.getSiblingsNumber() == t2.getSiblingsNumber())
579                 return new Pair((MathMLTree)t1.getParent(),
580                                 (MathMLTree)t2.getParent());
581
582         return new Pair(t1, t2);
583     }
584
585
586     private boolean containsIdenticalInCorpusTreeNodes(List<IdxNode<
            String>> idxNodelist,
587             IdxNode<String> CN) {
588
589         for(IdxNode<String> idxNode : idxNodelist)
590             if(idxNode.equals(CN))
591                 return true;
592
593         return false;
```

```
594     }
595
596
597     private String getIndexTableStringView(List<PRwithIdxNodes<String>>
            idxTbl) {
598
599         StringBuilder sb = new StringBuilder();
600
601         for(PRwithIdxNodes<String> cIdx : idxTbl)
602             sb.append(cIdx.getStringView()).append("\n");
603
604         return sb.toString();
605     }
606
607     private String getCorpusIndexTableStringView() {
608         return getIndexTableStringView(this.corpusIdxTable);
609     }
610
611     private String getQueryIndexTableStringView() {
612         return getIndexTableStringView(this.queryIdxTable);
613     }
614
615
616     public void printCorpusIndexTable() {
617
618         StringBuilder sb = new StringBuilder();
619         sb.append(this.getCorpusIndexTableStringView()).
620                 append("PR count: ").append(corpusIdxTable.size());
621
622         System.out.println(sb.toString());
623     }
624
625     public void printQueryIndexTable() {
626         StringBuilder sb = new StringBuilder();
627         sb.append(this.getQueryIndexTableStringView()).
628                 append("PR count: ").append(queryIdxTable.size());
629
630         System.out.println(sb.toString());
631     }
632
633
634
635     public int getNumberOfCommonNodesByNumCorpTree(int num) {
636
637         for(Sto sto: StoList)
638             if(sto.tr2.getTreeID() == num)
639                 return sto.numberCommonNodes();
640
641
642         System.out.println("Схожести с деревом T" + num + " не встретил
                ось");
643         return 0;
644     }
645
646
647 }
```

Листинг А.2. Исходный код класса IdxNode

```
1  package entities;
2
3  import additional.ConsoleStrings;
4  import java.util.Objects;
5
6
7  public class IdxNode<T> {
8
9      private final T label;
10
11     // идентификатор дерева, в котором нахожится этот узел
12     private final int treeID;
13
14     // каким по счету этот узел встретился в дереве
15     private final int repeatID;
16
17     // сколько листьев у поддерева, начинающегося с этого узла
18     private final int leavesCount;
19
20     // позиция в дереве с индексом treeID (в ширину)
21     private final int breadthPosInTree;
22
23     // те элементы, которые будут учитываться при сравнении и выводе на
           экран
24     public static enum Content {
25         label,
26         label_repeatID,
27         label_treeID_repeatID,
28         label_treeID_repeatID_leavesCount,
29         label_treeID_repeatID_leavesCount_breadthPosInTree,
30         label_leavesCount;
31     }
32
33     public IdxNode(T label, int treeID, int repeatID, int leavesCount,
           int breadthPosInTree) {
34         this.label = label;
35         this.treeID = treeID;
36         this.repeatID = repeatID;
37         this.leavesCount = leavesCount;
38         this.breadthPosInTree = breadthPosInTree;
39     }
40
41     public IdxNode(MathMLTree node) {
42
43         this.label = (T) node.getLabel();
44
45         if(!node.isRoot())
46             this.treeID = ((MathMLTree)node.getRoot()).getTreeID();
47         else
48             this.treeID = node.getTreeID();
49
50
51         if(node.isLeaf())
52             this.leavesCount = 0;
53         else
54             this.leavesCount = node.getLeafCount();
```

```
55
56
57          this.repeatID = node.getLabelIdxTime();
58          this.breadthPosInTree = node.getNodeBreadthOrder();
59      }
60
61      public IdxNode(IdxNode idxNode, int offsetBreadthOrder) {
62          this.label = (T) idxNode.getLabel();
63          this.treeID = idxNode.getTreeID();
64          this.leavesCount = idxNode.getLeavesCount();
65          this.repeatID = idxNode.repeatID;
66          this.breadthPosInTree = idxNode.getBreadthPosInTree() +
                offsetBreadthOrder;
67      }
68
69      public T getLabel() {
70          return this.label;
71      }
72
73      public int getTreeID() {
74          return this.treeID;
75      }
76
77      public int getRepeatID() {
78          return this.repeatID;
79      }
80
81      public int getLeavesCount() {
82          return this.leavesCount;
83      }
84
85      public int getBreadthPosInTree() {
86          return this.breadthPosInTree;
87      }
88
89      // узлы равны, если (все совпадает)
90      @Override
91      public boolean equals(Object obj) {
92
93          if(obj == this)
94              return true;
95
96          if(obj == null)
97              return false;
98
99          if (obj instanceof IdxNode<?>)
100             if(((IdxNode<?>)obj).label.equals(this.label) &&
101                     ((IdxNode<?>)obj).leavesCount == this.leavesCount
                            &&
102                     ((IdxNode<?>)obj).repeatID == this.repeatID &&
103                     ((IdxNode<?>)obj).treeID == this.treeID &&
104                     ((IdxNode<?>)obj).breadthPosInTree == this.
                            breadthPosInTree)
105                 return true;
106
107         return false;
108     }
109
```

```
110     public boolean equals(Content content, Object obj) {
111         if(obj == this)
112             return true;
113 
114         if(obj == null)
115             return false;
116 
117 
118         if (obj instanceof IdxNode<?>)
119             switch(content) {
120             case label:
121                 if(((IdxNode<?>)obj).label.equals(this.label))
122                     return true;
123 
124 
125             case label_treeID_repeatID:
126                 if(((IdxNode<?>)obj).label.equals(this.label) &&
127                    ((IdxNode<?>)obj).treeID == this.treeID &&
128                    ((IdxNode<?>)obj).repeatID == this.repeatID)
129                 return true;
130 
131 
132             case label_treeID_repeatID_leavesCount:
133                 if(((IdxNode<?>)obj).label.equals(this.label) &&
134                    ((IdxNode<?>)obj).treeID == this.treeID &&
135                    ((IdxNode<?>)obj).repeatID == this.repeatID &&
136                    ((IdxNode<?>)obj).leavesCount == this.leavesCount)
137                 return true;
138 
139 
140             case label_leavesCount:
141                 if(((IdxNode<?>)obj).label.equals(this.label) &&
142                    ((IdxNode<?>)obj).leavesCount == this.leavesCount)
143                 return true;
144         }
145 
146 
147         return false;
148     }
149 
150     // param - inColor -> выводить в цвете или нет
151     public String getStringView(Content viewstyle, boolean isInColor) {
152         StringBuilder sb = new StringBuilder();
153 
154         if(isInColor)
155             switch(this.treeID % 4) {
156                 case 1:
157                     sb.append(ConsoleStrings.ANSI_RED);
158                     break;
159                 case 2:
160                     sb.append(ConsoleStrings.ANSI_BLUE);
161                     break;
162                 case 3:
163                     sb.append(ConsoleStrings.ANSI_PURPLE);
164                     break;
165                 default:
166                     sb.append(ConsoleStrings.ANSI_BLACK);
167             }
```

```
168
169
170         sb.append(this.label.toString());
171
172         switch(viewstyle) {
173
174             case label_repeatID:
175                 sb.append(ConsoleStrings.LOWIDX).append(this.repeatID);
176                 break;
177
178             case label_treeID_repeatID:
179                 sb.append(ConsoleStrings.UPPERIDX).append(this.treeID);
180                 break;
181
182
183             case label_treeID_repeatID_leavesCount:
184                 sb.append(ConsoleStrings.SPACE).
185                 append("(").append(this.leavesCount).append(")");
186                 break;
187
188
189             case label_treeID_repeatID_leavesCount_breadthPosInTree:
190                 sb.append(ConsoleStrings.SPACE).
191                 append("(").append(this.leavesCount).append(")").append
                        (" (pos:").
192                             append(this.breadthPosInTree).append(")");
193         }
194
195         if(isInColor)
196             sb.append(ConsoleStrings.ANSI_RESET);
197
198         return sb.toString();
199     }
200
201     public void print(Content viewStyle, boolean isInColor) {
202         System.out.println(getStringView(viewStyle, isInColor));
203     }
204
205     @Override
206     public int hashCode() {
207         int hash = 3;
208         hash = 83 * hash + Objects.hashCode(this.label);
209         return hash;
210     }
211 }
```

Листинг А.3. Исходный код класса MathExpression

```
1  package entities;
2
3  import GUI.ResultFrame;
4  import java.io.File;
5  import java.io.FileReader;
6  import java.io.IOException;
7  import java.util.Scanner;
8  import java.util.logging.Level;
9  import java.util.logging.Logger;
```

```
10 import net.sourceforge.jeuclid.DOMBuilder;
11 import net.sourceforge.jeuclid.MathMLParserSupport;
12 import net.sourceforge.jeuclid.app.mathviewer.Messages;
13 import net.sourceforge.jeuclid.layout.LayoutableDocument;
14 import net.sourceforge.jeuclid.layout.LayoutableNode;
15 import net.sourceforge.jeuclid.swing.JMathComponent;
16 import org.w3c.dom.Document;
17 import org.w3c.dom.Node;
18 import org.xml.sax.SAXException;
19 import org.apache.commons.logging.Log;
20 import org.apache.commons.logging.LogFactory;
21
22
23
24 public class MathExpression {
25
26     // для отображении выражения в swing
27     private final JMathComponent jMathComponent;
28
29     // структурное дерево
30     private final MathMLTree mathMLTree;
31
32     // название файла / ресурса, откуда получено выражение
33     public final String fileName;
34
35
36     public MathExpression(String path, String fileName) throws
           SAXException, IOException {
37
38         File f = new File(path, fileName);
39         Document domDocument = MathMLParserSupport.parseFile(f);
40
41
42         this.fileName = f.getName();
43         this.jMathComponent = new JMathComponent();
44         this.jMathComponent.setDocument(domDocument);
45         this.mathMLTree = MathMLTree.fromNode(domDocument.
               getDocumentElement());
46
47         this.mathMLTree.normilizeTree();
48     }
49
50     public MathExpression(File f) throws SAXException, IOException {
51         Document domDocument = MathMLParserSupport.parseFile(f);
52
53         this.fileName = f.getName();
54         this.jMathComponent = new JMathComponent();
55         this.jMathComponent.setDocument(domDocument);
56         this.mathMLTree = MathMLTree.fromNode(domDocument.
               getDocumentElement());
57
58         this.mathMLTree.normilizeTree();
59     }
60
61     public JMathComponent getJMathComponent() {
62         return this.jMathComponent;
63     }
64
```

```
65
66      public void printTreeStructure() {
67          mathMLTree.prettyPrint();
68      }
69
70      public MathMLTree getMathMLTree() {
71          return mathMLTree;
72      }
73
74
75  }
```

Листинг А.4. Исходный код класса MathMLTree

```
1   package entities;
2
3   import additional.FormatUtilities;
4   import additional.ConsoleStrings;
5   import additional.Pair;
6   import java.util.ArrayList;
7   import java.util.Enumeration;
8   import java.util.LinkedHashMap;
9   import java.util.List;
10  import java.util.Map;
11  import java.util.Vector;
12  import javax.swing.tree.DefaultMutableTreeNode;
13  import javax.swing.tree.TreeNode;
14  import org.w3c.dom.Node;
15  import org.w3c.dom.NodeList;
16
17
18
19  public class MathMLTree extends DefaultMutableTreeNode {
20
21      private String label;
22      private int treeID;
23
24
25      public static MathMLTree fromMathMLTree(MathMLTree trueTree) {
26
27          MathMLTree clonedTree = new MathMLTree(trueTree.getLabel(),
                trueTree.getTreeID());
28
29          for(Enumeration e = trueTree.children(); e.hasMoreElements();)
                {
30              MathMLTree sibling = (MathMLTree)e.nextElement();
31              clonedTree.add(fromMathMLTree(sibling));
32          }
33
34          return clonedTree;
35      }
36
37      public MathMLTree(String label, int treeID) {
38          this.label = null;
39          this.treeID = -1;
40          this.label = label;
41          this.treeID = treeID;
```

```
42      }
43
44      public static MathMLTree fromNode(Node node) {
45
46          int treeID = -1;
47
48          if(MathMLPresentationMarkupElement.contains(node.getNodeName())
                ) {
49              MathMLTree tree = new MathMLTree(node.getNodeName(), treeID
                    );
50              if(MathMLPresentationMarkupElement.isToken(node.getNodeName
                    ()))
51                  tree.add(new MathMLTree(node.getTextContent(), treeID))
                        ;
52
53              NodeList childs = node.getChildNodes();
54              for (int i = 0; i < childs.getLength(); i++)
55                  if(childs.item(i).getNodeType() == Node.ELEMENT_NODE)
56                      tree.add(fromNode(childs.item(i)));
57
58            return tree;
59          }
60
61
62          return null;
63      }
64
65      public void setTreeID(int treeID) {
66          this.treeID = treeID;
67      }
68
69      public int getTreeID() {
70          return treeID;
71      }
72
73      public void normilizeTree() {
74          for(TreeNode node : getLeaves()) {
75              MathMLTree prnt = (MathMLTree) node.getParent();
76
77              if(prnt.getLabel().equals("mi")) {
78                  String name = ((MathMLTree)node).getLabel();
79                  switch(name) {
80                      case "sin":
81                          ((MathMLTree)node).setLabel("TRIGF");
82                          break;
83                      case "cos":
84                          ((MathMLTree)node).setLabel("TRIGF");
85                          break;
86                      case "tan":
87                          ((MathMLTree)node).setLabel("TRIGF");
88                          break;
89                      case "cot":
90                          ((MathMLTree)node).setLabel("TRIGF");
91                          break;
92                      default:
93                          ((MathMLTree)node).setLabel("VARNAME");
94                  }
95              }
```

```
96
97              if(prnt.getLabel().equals("mo")) {
98                  String name = ((MathMLTree)node).getLabel();
99                  switch(name) {
100                     case "+":
101                         ((MathMLTree)node).setLabel("PLUSMINUS");
102                         break;
103                     case "-":
104                         ((MathMLTree)node).setLabel("PLUSMINUS");
105                         break;
106                 }
107             }
108
109
110             if(prnt.getLabel().equals("mn"))
111                 ((MathMLTree)node).setLabel("NUMVAL");
112         }
113     }
114
115     public void setLabel(String label) {
116         this.label = label;
117     }
118
119     // какой по счету среди своих родственных узлов считая слево направ
          о
120     // счет начинается с единицы
121     public int getSiblingsNumber() {
122
123         int k = 1;
124         DefaultMutableTreeNode prevNode;
125
126         if(getPreviousSibling() != null) {
127             prevNode = this.getPreviousSibling();
128             k++;
129
130
131             while(prevNode.getPreviousSibling() != null) {
132                 prevNode = prevNode.getPreviousSibling();
133                 k++;
134             }
135         }
136
137         this.getSiblingCount();
138         return k;
139     }
140
141     public List<Pair<String, Integer>> getPostOrderLabeledPathToRoot()
          {
142
143         List<Pair<String, Integer>> path = new ArrayList<>();
144
145         MathMLTree tr = this;
146
147         path.add(new Pair(tr.label, tr.getSiblingsNumber()));
148
149
150         while(!tr.isRoot()) {
151             tr = (MathMLTree)tr.getParent();
```

```
152
153             path.add(new Pair(tr.label, tr.getSiblingsNumber()));
154         }
155
156         return path;
157     }
158
159
160
161     // какой это узел по счету таким названием при проходе в ширину от
            корня дерева
162     /*
163     Алгоритм нахождения номера узла и повторялся ли он:
164         1. Сохраняем путь до корня
165         2. От корня дерева начинаем искать (в ширину) элемент с тем же
166             названием.
167         3. Когда находим - сверяем пути до корня
168             если совпали -> сохраняем номер узла, repeat = 1
169             если не совпали -> repeat++ , ищем дальше
170     */
171     public int getLabelIdxTime() {
172         int repeat = 0;
173         List path = this.getPostOrderLabeledPathToRoot();
174         MathMLTree root = (MathMLTree)this.getRoot();
175
176
177
178         for (Enumeration e = root.breadthFirstEnumeration(); e.
                hasMoreElements();) {
179             MathMLTree node = (MathMLTree)e.nextElement();
180             if(node.getLabel().equals(this.label))
181                 if(path.equals(node.getPostOrderLabeledPathToRoot()))
182
183                     break;
184                 else
185                     repeat++;
186         }
187
188         return ++repeat;
189     }
190
191     // какой элемент по счету при проходе в ширину
192     // связан по логике с getLabelIdxTime()
193     public int getNodeBreadthOrder() {
194
195         int order = 0;
196         List path = this.getPostOrderLabeledPathToRoot();
197         MathMLTree root = (MathMLTree)this.getRoot();
198
199         for (Enumeration e = root.breadthFirstEnumeration(); e.
                hasMoreElements();) {
200             MathMLTree node = (MathMLTree)e.nextElement();
201             order++;
202             if(node.getLabel().equals(this.label))
203                 if(path.equals(node.getPostOrderLabeledPathToRoot()))
204                     break;
205         }
206
```

```
207         return order;
208     }
209
210     public void setTreeIDSameForAllTree(int trID) {
211
212         MathMLTree tr = this;
213
214         while(!tr.isRoot())
215             tr = (MathMLTree)tr.getParent();
216
217         for (Enumeration e = breadthFirstEnumeration(); e.
                hasMoreElements();) {
218             MathMLTree t = (MathMLTree) e.nextElement();
219             t.setTreeID(trID);
220         }
221
222     }
223
224     // дочерние узлы также наследуют treeID
225     public static MathMLTree fromString(String s) {
226
227         int treeID = FormatUtilities.getTreeID(s);
228
229         s = s.substring(s.indexOf("{"), s.lastIndexOf("}") + 1);
230         MathMLTree node = new MathMLTree(FormatUtilities.getRoot(s),
                treeID);
231
232         Vector c = FormatUtilities.getChildren(s);
233         for(int i = 0; i < c.size(); i++)
234             node.add(fromString((String)c.elementAt(i)));
235
236         return node;
237     }
238
239     public int getNodeCount(){
240         int sum = 1;
241         for(Enumeration e = children(); e.hasMoreElements();)
242             sum += ((MathMLTree)e.nextElement()).getNodeCount();
243
244         return sum;
245     }
246
247
248     public String getStringView(boolean showTreeID) {
249
250         StringBuilder sb = new StringBuilder();
251
252         for(int i = 0; i < getLevel(); i++)
253             sb.append(ConsoleStrings.TAB);
254
255
256         if(showTreeID)
257             sb.append("trID: ").append(this.treeID).append("\n");
258
259
260         if(!isRoot())
261             sb.append(ConsoleStrings.BRANCH);
262         else
```

```
263             sb.append(ConsoleStrings.ROOT);
264
265
266         sb.append(ConsoleStrings.SPACE).
267             append(ConsoleStrings.LEFTQUOTES).
268             append(this.label).
269             append(ConsoleStrings.RIGHTQUOTES).append("\n");
270
271         for(Enumeration e = children(); e.hasMoreElements();)
272                 sb.append(((MathMLTree)e.nextElement()).getStringView(
                        false));
273
274         return sb.toString();
275     }
276
277
278     public void print(boolean printTreeID) {
279         System.out.println(getStringView(printTreeID));
280     }
281
282     public void prettyPrint() {
283         for(int i = 0; i < getLevel(); i++)
284             System.out.print(ConsoleStrings.TAB);
285
286         if(!isRoot())
287             System.out.print(ConsoleStrings.BRANCH);
288         else {
289             System.out.println("trID: " + this.treeID);
290             System.out.print(ConsoleStrings.ROOT);
291         }
292
293         System.out.print((new StringBuilder(" '")).append(label).append
                ("' ").toString());
294         System.out.println();
295
296
297         for(Enumeration e = children(); e.hasMoreElements();
298                 ((MathMLTree)e.nextElement()).prettyPrint());
299     }
300
301     public String getLabel() {
302         return this.label;
303     }
304
305     public Vector<TreeNode> getLeaves() {
306
307         Vector<TreeNode> leaves = new Vector<>();
308
309
310         TreeNode node;
311         Enumeration enum_ = breadthFirstEnumeration(); // order matters
                not
312
313         while (enum_.hasMoreElements()) {
314             node = (TreeNode)enum_.nextElement();
315             if (node.isLeaf())
316                 leaves.add(node);
317         }
```

```
318
319         return leaves;
320     }
321
322     public TreeNode getNodeByBreadthOrderIndex(int n) {
323
324         int order = 0;
325         for (Enumeration e = breadthFirstEnumeration(); e.
                hasMoreElements();) {
326             MathMLTree t = (MathMLTree) e.nextElement();
327             if (++order == n)
328                 return t;
329         }
330
331         return null;
332     }
333
334
335 }
```

Листинг А.5. Исходный код класса ProductionRule

```
1  package entities;
2
3  import additional.ConsoleStrings;
4  import static entities.MathMLTree.fromNode;
5  import java.util.ArrayList;
6  import java.util.Collection;
7  import java.util.Collections;
8  import java.util.Enumeration;
9  import java.util.List;
10 import java.util.Objects;
11 import javax.swing.tree.TreeNode;
12 import org.w3c.dom.Node;
13 import org.w3c.dom.NodeList;
14
15
16 public final class ProductionRule<T> {
17
18     private final T topNode;
19     private final List<T> children;
20
21
22     public ProductionRule(T topNode, List<T> children) {
23         this.topNode = topNode;
24
25         if(children == null)
26             this.children = new ArrayList<>();
27         else
28             this.children = children;
29     }
30
31     public ProductionRule(TreeNode node) {
32         // T -> String
33
34         this.topNode = (T) ((MathMLTree)node).getLabel();
35         this.children = new ArrayList<>();
```

```
36
37          for(Enumeration e = node.children(); e.hasMoreElements();)
38              this.children.add((T) ((MathMLTree)e.nextElement()).
                    getLabel());
39      }
40
41
42      public List<T> getChildren() {
43          return Collections.unmodifiableList(this.children);
44      }
45
46      public T getTopNode() {
47          return this.topNode;
48      }
49
50      public String getStringView() {
51          StringBuilder sb = new StringBuilder();
52
53          sb.append(ConsoleStrings.LEFTQUOTES).
54                  append(topNode.toString()).
55                  append(ConsoleStrings.RIGHTQUOTES);
56
57          if(!children.isEmpty()) {
58
59              sb.append(ConsoleStrings.PR);
60              for (T child : children)
61                  sb.append(ConsoleStrings.SPACE).
62                          append(ConsoleStrings.LEFTQUOTES).
63                          append(child.toString()).
64                          append(ConsoleStrings.RIGHTQUOTES);
65          }
66
67          return sb.toString();
68      }
69
70      public void print() {
71          System.out.println(getStringView());
72      }
73
74      @Override
75      public boolean equals(Object obj) {
76
77          if(obj == this)
78              return true;
79
80          if(obj == null)
81              return false;
82
83          if (obj instanceof ProductionRule<?>)
84              if(((ProductionRule<?>)obj).topNode.equals(this.topNode) &&
85                      ((ProductionRule<?>)obj).children.equals(this.
                            children))
86                  return true;
87
88          return false;
89      }
90
91      @Override
```

```
92      public int hashCode() {
93          int hash = 7;
94          hash = 31 * hash + Objects.hashCode(this.topNode);
95          hash = 31 * hash + Objects.hashCode(this.children);
96          return hash;
97      }
98
99  }
```

Листинг А.6. Исходный код класса PRwithIdxNodes

```
1   package entities;
2
3   import additional.ConsoleStrings;
4   import java.util.ArrayList;
5   import java.util.List;
6
7
8   public class PRwithIdxNodes<T> {
9
10      private final ProductionRule<T> PR;
11      private List<IdxNode<T>> idxNodes;
12
13
14      public PRwithIdxNodes(ProductionRule<T> PR) {
15          this.PR = PR;
16          this.idxNodes = new ArrayList<>();
17      }
18
19      public void add_topCorpusTreeNode(IdxNode<T> node) {
20          this.idxNodes.add(node);
21      }
22
23      public String getStringView() {
24          StringBuilder sb = new StringBuilder();
25
26          sb.append(PR.getStringView());
27
28          if(!idxNodes.isEmpty()) {
29              sb.append(ConsoleStrings.DSPACE).
30                  append(ConsoleStrings.PR_TOPNODES_DELIMITER).append(
                        ConsoleStrings.DSPACE);
31              for(IdxNode<T> node : idxNodes)
32                  sb.append(node.getStringView(IdxNode.Content.
                        label_treeID_repeatID_leavesCount_breadthPosInTree
                        ,true)).
33                          append(ConsoleStrings.DSPACE);
34          }
35
36          return sb.toString();
37      }
38
39      public void print() {
40          System.out.println(getStringView());
41      }
42
43      public ProductionRule<T> getPR() {
```

```
44          return this.PR;
45      }
46
47      public List<IdxNode<T>> getIdxNodes() {
48          return this.idxNodes;
49      }ы
50  }
```

Листинг А.7. Исходный код класса Sto

```
1   package entities;
2
3   import java.text.DecimalFormat;
4
5
6   public class Sto {
7
8       public static enum Type {
9           SUBEXPRESSION,
10          STRUCTURAL,
11          IRREGULAR;
12      }
13
14      public final Type type;
15      public final MathMLTree tr1;
16      public final MathMLTree tr2;
17      public final MathMLTree commonPart;
18
19      // additional information
20      private final IdxNode<String> overlayIdxNode1;
21      private final IdxNode<String> overlayIdxNode2;
22      private final double numberCommonNodes;
23      private final double numberAllNodes;
24
25
26
27      public Sto(Type type,
28          MathMLTree tr1, IdxNode<String> overlayIdxNode1,
29          MathMLTree tr2, IdxNode<String> overlayIdxNode2,
30          int overlapCount) {
31
32
33          this.type = type;
34          this.tr1 = tr1;
35          this.tr2 = tr2;
36          this.commonPart = null;
37
38          // additional
39
40          this.overlayIdxNode1 = overlayIdxNode1;
41          this.overlayIdxNode2 = overlayIdxNode2;
42          switch(type) {
43              case SUBEXPRESSION:
44                  this.numberCommonNodes = overlapCount * 2;
45                  break;
46              default:
47                  this.numberCommonNodes = overlapCount * 2;
```

```
48          }
49
50          this.numberAllNodes = tr1.getNodeCount() + tr2.getNodeCount();
51      }
52
53      public Sto(Type type, MathMLTree tr1, MathMLTree tr2) {
54
55
56          this.type = type;
57          this.tr1 = tr1;
58          this.tr2 = tr2;
59          this.commonPart = null;
60
61          // additional
62          this.overlayIdxNode1 = null;
63          this.overlayIdxNode2 = null;
64          this.numberCommonNodes = 0;
65          this.numberAllNodes = tr1.getNodeCount() + tr2.getNodeCount();
66      }
67
68
69      public String simValueString() {
70          DecimalFormat formatter = new DecimalFormat("#0.000");
71
72          return formatter.format(this.simValue());
73      }
74
75      public String simNodesString() {
76
77          return ((int)numberCommonNodes + " / " + (int)numberAllNodes);
78      }
79
80      public double simValue() {
81          double Sim = numberCommonNodes / numberAllNodes;
82          return Sim;
83      }
84
85      public int numberCommonNodes() {
86          return (int) this.numberCommonNodes;
87      }
88
89
90      public static enum ViewStyle {
91          OVERLAYNODES,
92          COMMONTREEPART;
93      }
94
95      public String getStringView(ViewStyle style) {
96
97          StringBuilder sb = new StringBuilder();
98
99          switch(style) {
100             case OVERLAYNODES:
101                 sb.append("Sto(T").
102                    append(tr1.getTreeID()).append(",T").
103                    append(tr2.getTreeID()).append(") = ").
104                    append(simNodesString());
105
```

```
                if (overlayIdxNode1 != null && overlayIdxNode2 != null
                    && type == Type.SUBEXPRESSION)
                    sb.append(" (").append(overlayIdxNode1.getStringView
                        (IdxNode.Content.label, true)).append(",").
                    append(overlayIdxNode2.getStringView(IdxNode.Content
                        .label, true)).append(")");

                break;
            }

            return sb.toString();
        }


        public void print(ViewStyle style) {
            System.out.println(getStringView(style));
        }
}
```

Листинг А.8. Исходный код класса Pair

```
package additional;

import java.util.Objects;

 public class Pair<N, M> {
    public final N n;
    public final M m;

    public Pair(N n, M m) {
        this.n= n;
        this.m= m;
     }

    @Override
    public boolean equals(Object obj) {

        if(obj == this)
            return true;

        if(obj == null)
            return false;

        if (obj instanceof Pair<?,?>)
            if(n.equals(((Pair<?,?>)obj).n) && m.equals(((Pair<?,?>)obj
                ).m))
                return true;

        return false;
    }

    @Override
    public int hashCode() {
        int hash = 5;
        hash = 89 * hash + Objects.hashCode(this.n);
        hash = 89 * hash + Objects.hashCode(this.m);
        return hash;
```

```
36      }
37  }
```

Листинг А.9. Исходный код класса ConsoleStrings

```
1  package additional;
2
3  public final class ConsoleStrings {
4      public static final String SPACE = " ";
5      public static final String DSPACE = "  ";
6      public static final String TAB = "    ";
7
8      public static final String ROOT = "*---+";
9      public static final String BRANCH = "+---+";
10     public static final String PR = "->";
11
12     public static final String UPPERIDX = "^";
13     public static final String LOWIDX = "_";
14
15     public static final String ELEMENT_DELIMITER = ",";
16     public static final String PR_TOPNODES_DELIMITER = ":";
17
18     public static final String LEFTQUOTES = "'";
19     public static final String RIGHTQUOTES = "'";
20
21     public static final String ANSI_RESET = "\u001B[0m";
22     public static final String ANSI_BLACK = "\u001B[30m";
23     public static final String ANSI_RED = "\u001B[31m";
24     public static final String ANSI_GREEN = "\u001B[32m";
25     public static final String ANSI_YELLOW = "\u001B[33m";
26     public static final String ANSI_BLUE = "\u001B[34m";
27     public static final String ANSI_PURPLE = "\u001B[35m";
28     public static final String ANSI_CYAN = "\u001B[36m";
29     public static final String ANSI_WHITE = "\u001B[37m";
30 }
```