Филиал Московского Государственного Университета
имени М.В. Ломоносова в городе Ташкенте
Факультет прикладной математики и информатики
Кафедра прикладной математики и информатики

Бутузов Игорь Владимирович

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

НА ТЕМУ: «Поиск математических утверждений в базе данных» ПО НАПРАВЛЕНИЮ 01.04.02 «Прикладная математика и информатика»

ВКР рассмотрена и рекомендована к защите
зав. кафедрой «ПМиИ» д.ф.-м.н., профессор _____Кудрявцев В.Б.

Научный руководитель
к.ф.-м.н. _____Калачев Г.В.

«____» _____ 2020 год

Ташкент 2020

Аннотация

Данная работа посвящена разработке программы, способной найти само утверждение или похожее на него в базе данных с математическими теоремами. Такая программа сможет по запросам пользователя ранжировать теоремы таким образом, чтобы теоремы, максимально подходящие под запрос, получали более высокий приоритет при выдаче результата.

В первой части работы показаны результаты работы модуля тестирования. Для этой цели использовался набор из 8 тестовых теорем.

Во второй главе дано описание алгоритмов, которые используются для сравнения математических формул.

В третьей главе работы дано описание работы, реализованной программы. Рассматриваются этапы её работы, описываются алгоритмы работы основных модулей.

Ключевые слова: синтаксический анализ, теорема, функция ранжирования

Abstract

The first part of the paper shows the results of testing module. For this purpose, a set of 8 tested theorems was used.

The second chapter describes the algorithms that are used to compare mathematical formulas.

The third chapter of the paper describes the work of the implemented program. The stages of its work are considered, the algorithms of the main modules are described.

Keywords: syntax analysis, theorem, ranking function

Содержание

Введение

Математические теоремы обладают уникальной структурой синтаксиса и большим разнообразием семантически эквивалентных конструкций. Одну и ту же теорему можно записать несколькими способами. Например, изменив обозначення в используемых формулах на эквивалентные или изменив конструкцию, записанную на естественном языке, на конструкцию с тем же смыслом. По этой причине поиск по ключевым словам в теоремах может не дать желаемого результата. Если рассматривать только поиск по формулам, входящим в состав математического утверждения, не учитывая текст на естественном языке, то такой поиск имеет ряд существенных недостатков:

- Формула может встречаться в довольно большом количестве теорем;
- Формула может быть записана при помощи других обозначений или другой формулой с тем же семантическим значением;
- Формула может быть составной частью другой формулы;

Формализация структуры математического утверждения способна повысить вероятность найти искомое утверждение в базе данных с математическими утверждениями. Для формализации текста теорем на естественном языке удобно использовать язык логики предикатов. Перевод расплывчатой словесной формулировки на строгий, не допускающий противоречивых толкований язык логики предикатов способствует четкости и ясности мышления, но в процессе осуществления такого перевода возникает ряд трудностей. Например, такой процесс не может быть полностью автоматизирован, т.к. на результат работы программы может сильно изменится из-за одного неверно истолкованного слова.

Полученное представление текста теоремы на языке логики предикатов является аналогом формулы и оно может быть представлено в виде абстрактного дерева. Объединив такое дерево с формулами из данного утверждения, предварительно представленными в виде деревьев со схожей структуры, можно получить дерево исходного математического утверждения. К такому абстрактному дереву удобно применять алгоритмы работы с деревьями.

Результат перевода на язык логики предикатов будет более качественным, если входной текст для такого перевода будет структурирован и в нем будет содержаться вся необходимая для перевода информация [2],[6]. Такое представление текста могут дать синтаксические анализаторы. Задачей синтаксического анализатора является определение того какие члены предложения являются главными, а какие – подчиненными. Для этого у компьютера на входе есть цепочка символов, которую нужно правильно проинтерпретировать, разбить на слова, связать их между собой и построить синтаксическое дерево.

В данной работе используется синтаксический анализатор Stamford NLP [8], для работы которого используется модель СинТагРус [10], [11]. Для преобразования математических формул в форму дерева, используется РБНФ-грамматика (расширенная форма Бэкуса — Наура [7]) и библиотека для Python 3 TatSu [9], которая позволяет по файлу, содержащему правила построения грамматики, построить дерево формулы.

Постановка задачи

Целью данной работы является создание программы, способной найти само утверждение или похожее на него в базе данных с математическими теоремами.

Данная система сможет по запросам пользователя ранжировать теоремы таким образом, чтобы теоремы, максимально подходящие под запрос, получали более высокий приоритет при выдаче результата.

Актуальность данной темы заключается в том, что ежегодно доказывается несколько сотен тысяч теорем и других математических утверждений. Поиск нужного утверждения в столь огромном массиве данных представляется нетривиальной задачей.

В отличие от поиска по ключевым словам реализуемая система будет иметь более высокий результат, так как будет опираться не на вхождение определенных слов в запрос, а на смысловое содержание введенной теоремы. Поэтому такая система математического поиска сможет сопоставлять теорему, которую ввел пользователь и теорему из базы данных, даже если они записаны при помощи разных формулировок.

Предполагается, что как заданная теорема, так и теоремы из базы данных записаны на естественном языке (возможно, с использованием математических формул, представленных в формате системы TeX). При этом в программе должно учитываться, что формулировки заданной теоремы и той же теоремы в базе данных могут несколько отличаться.

В процессе работы было решено несколько промежуточных задач:

1. Задача перевода текста, написанного на естественном языке, на язык формул логики предикатов (с помощью синтаксического и семантического анализа), и последующего представления этих формул в виде деревьев;

2. Задача представления в том же виде формул, записанных в формате системы TeX;

3. Задача ранжирования формул базы данных, записанных в виде деревьев, по степени похожести на формулу теоремы-запроса.

Кроме того, для проверки работоспособности программы необходима база теорем, а также некоторое множество теорем, выступающих в качестве запросов к программе.

В качестве входных данных для данной программы используется синтаксический анализ Stamford. Программа реализована на языке Python 3. Результатом работы программы является такое представление базы математических утверждений, в которых формулировка искомой теоремы или похожей на нее должна появиться среди лучших кандидатов, найденных программой.

Сформулируем основные понятия, используемые в данной работе:

Токен — слово, устойчивое выражение или знак препинания. Токен имеет словоформу, лемму и набор морфологических характеристик.

Синтаксическое дерево зависимостей — способ представления синтаксической структуры предложения. В узлах дерева расположены токены данного предложения [3]. Под словами «родительский» и «дочерний» по отношению к токенам в данной работе следует понимать отношения между соответствующими узлами дерева. Каждому токену (исключение составляет корень синтаксического дерева) поставлено в соответствие положительное число – номер другого токена, который является родителем для данного. Для корня синтаксического дерева это число равно нулю. Между «родительским» и «дочерним» токенами устанавливается синтаксическое отношение, его мы будем называть типом связи. У корня синтаксического дерева родителя нет.

Словоформа — форма слова, полученная из основной части слова с помощью склонения и спряжения.

Лемма — словарная форма слова.

1 Полученные результаты

Результатом данной работы является реализованная программа, которая способна найти само утверждение или похожее на него в базе данных с математическими теоремами.

Данная программа может по запросам пользователя ранжировать теоремы таким образом, чтобы теоремы, максимально подходящие под запрос, получали более высокий приоритет при выдаче результата.

Для тестирования качества работы данной программы было написан специальный модуль тестирования. Тестировалась работа программы, начиная от поступления математической теоремы (в виде утверждения или таблицы с синтаксической и морфологической информацией) и заканчивая работой модуля отвечающего за ранжирование формул из базы данных .

Для тестирования каждой теореме присваивался свой уникальный номер. Номера формулировок одних и тех же теорем объединялись в некоторые множества номеров. Допустим, нам известно, что формулировки теоремы Больцано-Коши имеют номера 1,2. Соответственно, множество номеров для данной теоремы $\{1, 2\}$

1. Для каждой теоремы из файла, содержащего утверждения теорем строилась своя формула и данная формула заносилась в базу данных. Таким образом, строилась база формул с которой и происходило сравнение заданной формулы.

2. Из каждой теоремы из базы данных по ее утверждению или массиву с синтаксической информацией повторно строилось ее представление ее в виде формулы.

3. Данная формула сравнивалась с остальными формулами базы данных. Таким образом, строилась таблица ранжирования для данной формулы.

4. Каждый индекс формулы в таблице ранжирования сравнивался с множеством номеров, содержащих номер заданной теоремы. Если текущий индекс формулы из таблицы ранжирования совпал с некоторым элементом множества номеров, то данный элемент исключался из дальнейшего рассмотрения. Если текущего индекса нет в массиве номеров, то накладывался штраф, равный -1. Чем меньше штрафов было получено, тем лучше качества ранжирования.

Опишем алгоритм более подробно. Имеется база данных формул в количестве 8 элементов. A_i $(i = 1..8)$. Для каждого i известны индексы той же самой теоремы, но в другой формулировке. Предположим, что для теоремы с номером i теоремой с другой формулировкой будет теорема с индексом $j\,(j \neq i,\, j = 1..8)$. Для данной теоремы множество номеров равно $\{i, j\}$. Выберем произвольный номер k $(k = 1..8)$. Для теоремы A_k из изначальной базы данных проведем все этапы работы программы. Сравним разбор теоремы A_k с каждой имеющейся в базе данных формул A_i. Получим таблицу ранжирования с индексами формул $p_1, p_2, ..., p_8$. Итоговый результат будем хранить в переменной $mark$.

$mark = 0$. Для $\forall m$ $(m = 1..8)$:

Если $p_m \neq i$ && $p_m \neq j$, то $mark = mark - 1$

Если $p_m == i$, то множество номеров будет равно $\{j\}$

Если $p_m == j$, то множество номеров будет равно $\{i\}$

Если множество номеров пусто, то завершаем тестирование.

Для полученного значения $mark$ необходимо провести нормирование, чтобы его значение находилось между 0 и 1.

$$Result = 1 - \frac{mark}{size(A)}$$

Для тестовой выборки из 8 теорем были получены следующие результаты:

№ алгоритма	1	2	3	4	5	6	7	8
1	0.75	0.62	0.62	0.88	0.62	0.62	0.25	0.25
2	0.62	0.38	1.0	0.88	0.88	0.88	0.24	0.25

Полученные результаты можно трактовать следующим образом: второй алгоритм показал более высокий результат для Теоремы Ролля, однако качество его работы уступает первому алгоритму для Второй теоремы Больцано-Коши. При разборе Формулы Ньютона-Лейбница оба алгоритма показали одинаковый, тем не менее не высокий результат.

Теоремы классифицировались следующим образом:
Вторая теорема Больцано Коши: 1, 2;
Теорема Ролля: 3,4,5,6
Формула Ньютона-Лейбница: 7, 8;

1. Если непрерывная функция, определённая на вещественном интервале, принимает два значения, то она принимает и любое значение между ними.

2. Пусть функция f непрерывна на отрезке $[a, b]$, причем $f(a) \neq f(b)$, тогда для любого числа C, заключенного между $f(a)$ и $f(b)$, найдется точка $\gamma \in (a, b)$, что $f(\gamma) = C$.

3. Если вещественная функция, непрерывная на отрезке $[a, b]$ и дифференцируемая на интервале (a, b) , принимает на концах отрезка $[a, b]$ одинаковые значения, то на интервале (a, b) найдётся хотя бы одна точка, в которой производная функции равна нулю.

4. Пусть функция дифференцируема в открытом промежутке , на концах этого промежутка сохраняет непрерывность и принимает одинаковые значения: $f(a) = f(b)$, тогда существует точка c , в которой производная функции равна нулю : $f'(c) = 0$.

5. Пусть функция $f(x)$ непрерывна на отрезке $[a, b]$ и дифференцируема на интервале (a, b), причем $f(a) = f(b)$, тогда существует точка $c \in [a, b]$ такая, что $f'(c) = 0$.

6. Если функция f непрерывна на $[a, b]$, функция f дифференцируема во всех внутренних точках $[a, b]$ и $f(a) = f(b)$, тогда существует точка $c \in (a, b)$, в которой $f'(c) = 0$.

7. Если $f(x)$ непрерывна на отрезке $[a, b]$ и $\Phi(x)$ — любая её первообразная на этом отрезке, то имеет место равенство $\int\limits_a^b f(x)\,dx = \Phi(b) - \Phi(a) = \Phi(x)\Big|_a^b$.

8. Если $F(x)$ – любая первообразная функции $f(x)$, то справедливо равенство

$$\int\limits_a^x f(t)\,dt = F(x) - F(a)$$

.

Также модуль тестирования позволяет применять функцию ранжирования для тестирования качества разбора. Вручную были разобраны 15 теорем. Утверждения этих теорем подавались на вход программе разбора. Функция ранжирования выдавала схожесть между двумя разборами. Верхнее значение результат до подставления формулы в разбор, нижнее после.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.21	0.23	0.32	0.24	0.56	0.46	0.37	0.09	0.31	0.24	0.35	0.45	0.4	0.11	0.24
0.38	0.33	0.39	0.34	0.56	1.0	0.73	1.0	0.42	0.23	0.35	0.57	0.4	0.8	0.53

2 Обзор существующих алгоритмов

Поиск по формулам, входящим в состав математического утверждения, без учета текста, записанного на естественном языке имеет ряд существенных недостатков: например, формула может встречаться в нескольких теоремах, она может быть записана при помощи других обозначений или другой формулой с тем же семантическим значением.

Однако алгоритмы, используемые при таком поиске, могут быть востребованы, если получится представить математическое утверждение вместе с формулой и текстом на естественном языке в виде некоторой обобщенной формулы.

В работе [4] исследуется структурно-синтаксическое сходство математических выражений. Основное внимание уделяется модификации алгоритма наложения деревьев зависимостей для математических выражений, представленных в MathML. Рассмотрено применение трех алгоритмов.

В данной работе описано применение алгоритма Tree edit distance (Расстояние редактирования дерева) для математических выражений. Данный алгоритм определяет схожесть между двумя деревьями как число операций добавления, удаления и изменения для того, чтобы преобразовать одно дерево в другое.

Также в данной работе рассмотрен алгоритм Subpath set (Множество подпутей). Подпутем называется множество всех путей из корневой вершины к остальным вершинам, включая промежуточные пути. Коэффициент схожести в данном алгоритме определяется как число общих подпутей у рассматриваемых деревьев.

Третим представленным алгоритмом является модификация Tree overlapping algorithm (Алгоритм наложения деревьев). Данный алгоритм может быть описан следующим образом: при наложении произвольной вершины n_1 дерева T_1 на вершину n_2 дерева T_2 может возникнуть одинаковое продукционное правило наложения деревьев T_1 и T_2. Под схожестью в данном алгоритме понимается количество таких продукционных правил.

В работе [5] описан подход сравнения двух математических формул, представленных в MathML, при помощи чередования прямого и обратного обходов деревьев.

3 Описание системы математического поиска

3.1 Описание графического интерфейса программы

Для удобства использования всех модулей программы написан графический интерфейс. Интерфейс представляет собой 4 текстовых поля и ряд кнопок:

Текстовые поля

1.1. Поле ввода используется для ввода информации.

1.2. Два поля вывода данных нужны для вывода результата работы.

1.3. Поле вывода ошибок используется для вывода ошибок, возникших в результате работы.

Кнопки

2.1. Clean - очистка всех текстовых полей кроме поля ввода.

2.2. Load_and_run. После нажатия данной кнопки произойдет обращение к файлу конфигурации, будет найдено поле Infile. К соответствующему этому полю путю будет добавлен путь корневой директории . Данные из файла, находящегося по данному путю (в качестве данных могут выступать текстовое утверждение или массив с синтаксической информацией) будут преобразованы в формулу данного утверждения. Результат разбора будет выведен в первое поле вывода данных. Входные данные для разбора будут выведены во второе поле вывода данных.

2.3. Parse - данные из поля ввода (текстовое утверждение или массив с синтаксической информацией) будут преобразованы в формулу данного утверждения. Результат разбора будет выведен в первое Поле вывода данных.

2.4. Rank - запуск ранжирования. Данные из поля ввода (текстовое утверждение или массив с синтаксической информацией) будут преобразованы в соответствующую формулу данного утверждения. Полученная формула будет сравниваться с каждой формулой из базы формул. Чем выше результат сравнения (от 0 до 1), тем выше формула из базы формул будет в итоговой выдаче. Отсортированные формулы из базы формул будут выведены в первое поле вывода данных. Первым будет выводится разбор данных из поля ввода. Результаты сравнения будут выведены во второе поле вывода данных.

2.5. Test - запуск тестирующего модуля.

2.6. Picture - модуль отрисовки формул. Входные данные должны быть в формате списка вложенных списков. Данный модуль отрисовывает данные из поля ввода в отдельном окне.

2.7. Edit_config - редактирование файла конфигурации. Данные из данного файла будут выведены во второе поле вывода данных. Для сохранения изменений изменений необходимо нажать кнопку Save_config.

2.8. Help - вывод вспомогательной информации.

3.2 Описание базы данных с математическими утверждениями

Для тестирования работы реализованной программы были выбраны 300 теорем на русском языке из Википедии и различных учебных пособий. Теоремы были взяты из различных разделов математики.

Предварительно было произведено разделение текстовой части утверждения теорем и содержащихся в них математических формул на языке TeX. Формулы добавлялись в отдельный текстовый файл, а в самих теоремах заменялись на специальные обозначения вида formula_№, где № – порядковый номер формулы в данном файле.

К каждому преобразованному таким образом математическому утверждению применялся синтаксический анализатор Stamford NLP, который создавал таблицу для каждого слова со следующими полями:

№ Слово Лемма Номер родителя Тип синтаксической связи Часть речи

В полученную таблицу вносился ряд правок. Например, добавление недостающих обозначений для математических объектов, замена местоимений вида «она»,«он» на слова, которые они заменяли. Также слова из словаря заменялись на свои латинские эквиваленты. Несколько подряд идущих токенов, которые образовывали конструкцию («имеет место»,«интегрируема по Риману»), объединялись в одну конструкции.

1. Данные из таблицы преобразовывались в абстрактное синтаксическое дерево. Абстрактным синтаксическим деревом в данной программе является список вложенных списков, где заголовком каждого списка является родительский токен.

2. По полученному абстрактному дереву строилось множество всех возможных путей из корневой вершины. Из данного множества строятся выражения, которые затем и составят формулу логики предикатов.

Для преобразования математических формул в абстрактное синтаксическое дерево, используется РБНФ-грамматика (расширенная форма Бэкуса — Наура) и библиотека для Python 3 TatSu, которая позволяет по файлу с правилами грамматики построить дерево.

3. Специальные обозначения, введенные ранее в формулах логики предикатов, заменялись на деревья соответствующих формул. Полученная формула логики предикатов виде списка вложенных списков вносилась в текстовый файл.

3.3 Описание модуля перевода текста, написанного на естественном языке, на язык формул логики предикатов

Входными данными для данного модуля являются предложения, записанные на естественном языке. Математические формулы в данном предложении заменены на обозначения типа «formula_№», где № – номер соответствующего обозначения или формулы в файле со списком обозначений и формул. Выходными данными для данного модуля является исходное предложение, записанное при помощи логики предикатов.

Входные данные подаются синтаксическому анализатору Stamford NLP. Результат работы анализатора представляет собой таблицу (массив), содержащий подробную морфологическую и синтаксическую информацию, соответствующую обозначениям Национального Корпуса Русского Языка (СинТагРус) [11]. Каждый токен входного предложения представляет собой отдельный массив в данной таблице. В данном массиве содержится порядковый номер токена, его словоформа, лемма, номер родителя, тип связи между родителем и данным токеном, часть речи данного токена.

Для дальнейшей работы в полученную таблицу необходимо внести корректировки.

На первом этапе несколько отдельных токенов объединяются в один и заменяются либо на соответствующее название на английском языке, либо на стандартное математическое обозначение. Несколько подряд идущих токенов в таблице токенов. Например, токены, образующие конструкцию вида «интегрируема по Риману», будут заменены одним токеном R. Конструкция «равномерно непрерывна» будет заменена на конструкцию «uniform_contin», а конструкция «тогда и только тогда, когда» будет заменена на токен «<=>». При каждой такой замене происходит пересчет номеров родителей для дочерних токенов.

Для математических терминов написан словарь, где указаны соответствующая лемма термина и эквивалентное ему слово, на которое его можно заменить. Если произошло совпадение токена и значения в словаре, то будет произведена замена. Также на этом этапе к некоторым токенам при необходимости добавляются соответствующие обозначения. Стандартное обозначение будет добавлено для объектов «точка», «функция», «множество», «последовательность» при отсутствии у них уже существующего обозначения.

Если добавляемое стандартное обозначение уже встретилось в программе и используется для другого объекта, то будет к стандартному обозначению добавляется номер.

Также на предварительном этапе слова вида «его», «эта» заменяются на обозначение термина, который подразумевается под данным словом. Для данного типа токенов смотрится следующий за ним элемент, его значение запоминается и затем осуществляется поиск другого токена с такой же словоформой. Если таких токенов несколько, то будет отдан приоритет тому токену, который имеет такую же синтаксическую связь как у токена, который был запомнен. Если поиск удачен, то среди дочерних элементов найденного токена ищется его обозначение. Данное обозначение подставляется вместо «его», «эта».

Слова не имеющие смысловой важности, такие как «неравенство», «условие» могут быть удалены из таблицы токенов при условии, что следующий за ними токен является обозначением и это обозначение позволяет однозначно идентифицировать слово, которое предполагается удалить. Если удаление допустимо, предварительно все дочерние связи данных слов переносятся на обозначение данного слова. Например, для конструкции «выполнено неравенство formula_4», где родителем токена «неравенство» является токен «выполнено», необходимо проверить, что formula_4 однозначно идентифицировать неравенство. После корректировки родителем токена formula_4 станет токен «выполнено», formula_4 получит тот же тип связи, который связывал «неравенство» и «выполнить». После этого токен «неравенство» будет удален.

№	Название	Эквивалент в программе	Добавляемое обозначение
1	Функция	function	f
2	Множество	set	a
3	Отрезок	closed_interval	-
4	Непрерывный	C (continuity)	-
5	Монотонный	M (monotonic)	-

Таблица 1: Примеры заменяемых слов и добавляемых обозначений

Если в таблице с синтаксическим разбором есть «,» и после нее следуют специальные слова «a»,«то»,«тогда», то данная таблица будет разделена по этой запятой на две части с соответствующим пересчетом синтаксических связей. Следующий этап для каждой из частей будет проходить отдельно, а затем добавлены во временный массив. На следующем этапе данные из таблицы преобразовываются в абстрактное синтаксическое дерево (список вложенных списков). Используя номера токенов и их родителей рекурсивно строится такое представление дерева, в котором номер родителя является заголовком списка. Затем в полученном дереве номера токенов заменяется на конструкцию вида «лемма : тип синтаксического отношения».

Затем из полученных данных строится множество всевозможных путей из корневой вершины. Будем называть такое множество - множеством путей.

Опишем некоторые преобразования из данного модуля:

1. Из множества путей удаляются элементы, которые полностью содержатся в других.

2. Если элемент множества путей без специального слова («if»,«then») полностью совпадает с другим элементом этого множества, то оставляется только специальное слово. Иначе, происходит разделение элемента множества на два: специальное слово и остальную часть.

3. Если 2 элемента множества путей начинаются с одинакового слова и не входят в список объектов («функция»,«точка»), то эти два элемента объединяется.

4. Сочетания объект и соответствующее определение, объект и обозначение добавляются в множество путей.

5. Для каждого из специальных слов (кванторов) «if»,«then»,«forall»,«exists» создается выражение для данного слова. Чтобы элемент множества путей вошел в данное выражение он должен либо содержать переменную (обозначение) на которую распространяется

действие квантора (это обозначение следует за кванторов), либо не содержать никак специальных слов, тогда оно будет отнесено в область действий предыдущего квантора. Если количество элементов в полученном выражении больше одного, то в заголовок данного выражения будет добавлена «&».

3.4 Описание модуля преобразования математических формул

На вход данному модулю поступает математическая формула, записанная в формате системы TeX. На предварительном этапе работы модуля при помощи регулярных выражений из нее удаляются слова, которые не несут никакой смысловой нагрузки: displaystyle, textstyle, right, left, limits, nolimits, quad. Некоторые обозначения заменяются на эквиваленты.

№	Обозначение	Эквивалент в программе	Значение
1	{ .. }	set	множество
2	[..]	closed_interval	отрезок
3	|| .. ||	norm	норма
4	| .. |	abs	модуль

Таблица 2: Примеры заменяемых обозначений

Для построения абстрактного синтаксического дерева формулы в данном модуле используется библиотека TatSu . Это библиотека генерирует код на языке Python из грамматик в вариации РБНФ (расширенная форма Бэкуса — Наура).

Описание грамматики в РБНФ представляет собой набор правил, определяющих отношения между терминальными символами (терминалами) и нетерминальными символами (нетерминалами).

Терминальные символы — это минимальные элементы грамматики, не имеющие собственной грамматической структуры. В РБНФ терминальные символы — это либо предопределённые идентификаторы (имена, считающиеся заданными для данного описания грамматики), либо цепочки — последовательности символов в кавычках или апострофах. Нетерминальные символы — это элементы грамматики, имеющие собственные имена и структуру. Каждый нетерминальный символ состоит из одного или более терминальных и/или нетерминальных символов, сочетание которых определяется правилами грамматики. В РБНФ каждый нетерминальный символ имеет имя, которое представляет собой строку символов.

Основным преимуществом парсера TatSu является то, что каждая операция парсинга грамматики при необходимости может быть отредактирована при помощи написания функции на языке Python 3.

3.5 Модуль функции ранжирования

В данной работе сравнивались два алгоритма ранжирования. Под коэффициентом ранжирования будем понимать некое число от 0 до 1. Чем это число ближе к 1, тем выше схожесть между двумя объектами.

В качестве входных данных для данного модуля поступают два дерева (формула), разобранных при помощи модуля перевода текста, написанного на естественном языке, на язык формул логики предикатов и модуля преобразования математических формул A и B. Для каждой из формул рекурсивно построим множество всех путей, начиная от корневой. Назовем эти два множества A' и B'.

Опишем первый алгоритм.

Для множеств A' и B' вычисляется коэффициент Жаккара по следующей формуле:

$$K = \frac{n(A' \cap B')}{n(A' \cup B')}, \text{где } n() \text{ – количество элементов в множестве } C$$

Если обе формулы одинаковы и отличаются лишь обозначениями для переменных, то коэффициент K равен 1. Чем больше различий между формулами, тем значение K ближе к 0.

Теперь опишем второй алгоритм.

Для всех элементов множества A' ищем идентичную конструкцию в B'. Идентичной конструкцией при сравнении двух множеств будем называть такое множество, которое может отличаться от того множества, с которым оно сравнивается только переобозначением переменных. За каждую совпадение увеличиваем значение функции c.

Затем для всех пар элементов A'_i и B'_j вычисляется коэффициент Жаккара $K_{a,b}$ по следующим формулам:

$$K_i = \frac{n(A'_i \cap B'_j)}{n(A'_i \cup B'_j)}$$

$$K_{a,b} = \sum_{i=1}^{n(A)} \frac{c \cdot K_i \cdot i}{n(A)},$$

где i, j – индексы, указывающий насколько далеко ушли от вершины формулы A,B соответственно; $n(A)$ – количество элементов в множестве A .

Повторим процедуру поменяв A'_i и B'_j местами в качестве входных параметров функции. Получим коэффициент Жаккара $K_{b,a}$. Полученные результаты $K_{a,b}$ и $K_{b,a}$ могут не находиться в пределах от 0 до 1. Поэтому проведем их нормировку. Итого, окончательное значение коэффициента ранжирования:

$$\text{Коэффициент ранжирования} = \min\left(\frac{K_{a,b}}{K_{a,a}}, \frac{K_{b,a}}{K_{b,b}}\right)$$

Заключение

Результаты, полученные в ходе выполнения проекта, можно резюмировать следующим образом:

1. Написана и протестирована программа, способная найти само утверждение или похожее на него в базе данных с математическими теоремами.

Данная работа весьма актуальна, потому что система, способная искать похожие математические утверждения в базах данных, написанные с использованием естественного языка и формул на языке TeX, сможет найти применение в любой сфере деятельности, где используется математический язык.

2. Для тестирования работы программы были выбраны 8 математических теорем из разделов математики. При тестировании результат проверялось сможет ли реализованная программа найти не только саму себя, но и другую формулировку этой же теоремы.

Несмотря на низкий результат программу нужно улучшать для разбора большего количества теорем, пополнять внутренние словари, повышать качество их разбора. Также дополнительный интерес представляет дальнейшее совершенствование алгоритма ранжирования.

Список литературы

[1] Игошин В.И. Математическая логика и теория алгоритмов. Москва. Издательский центр «Академия» 2008. Страница 196.

[2] Hiroyuki Yamauchi. Processing of syntax and semantics of natural language by Predicate Logic. Institute of Space and Aeronautical Science, University of Tokyo. https://www.aclweb.org/anthology/C80-1059
Страница 390

[3] *Большакова Е.И., Воронцов К.В., Ефремова Н.Э., Клышинский Э.С., Лукашевич Н.В., Сапин А.С.* Автоматическая обработка текстов на естественном языке и анализ данных: учеб. пособие — М.: Изд-во НИУ ВШЭ, 2017. — 269 с. Страницы 25-28.

[4] Evgeny Pyshkin. University of Aizu, Japan. Mathematical Equation Structural Syntactical Similarity Patterns: A TreeOverlapping Algorithm and Its Evaluation https://pdfs.semanticscholar.org/53d4/e6043e49e42cbd90fc86185a6c7af6030bbd.pdf

[5] Yuping Qin, Junnan Guo, Aihua Zhang. A Mathematical Formula Matching Algorithm Based on MathML https://www.atlantis-press.com/proceedings/lemcs-15/25838389

[6] Julian E. Herwitz. Natural Language to Predicate Logic https://www.cs.rochester.edu/~brown/173/exercises/samples/ParseTranslate10_2.pdf.
Страница 5.

[7] https://www.cl.cam.ac.uk/ mgk25/iso-14977.pdf

[8] https://stanfordnlp.github.io/stanfordnlp/

[9] https://tatsu.readthedocs.io/en/stable/intro.html

[10] https://tatsu.readthedocs.io/en/stable/intro.html

[11] http://www.ruscorpora.ru/new/instruction-syntax.html