УДК 004

НЕКОТОРЫЕ ПРОГРАММНЫЕ ИНСТРУМЕНТЫ ДЛЯ АВТОМАТИЗИРОВАННОГО ПОПОЛНЕНИЯ ТЕРМИНОЛОГИЧЕСКОГО СЛОВАРЯ ПРЕДМЕТНОЙ ОБЛАСТИ

[1]Р. А. Румянцев, [1,2]О. А. Невзорова

[1]Институт вычислительной математики и информационных технологий, Казанский (Приволжский) федеральный университет, г. Казань, ул. Кремлевская, д. 35, 420008

[2]Институт прикладной семиотики Академии наук Республики Татарстан, г. Казань, ул. Лево-Булачная, 36А, 420111

romanrar93@gmail.com, onevzoro@gmail.com

Аннотация

Описано приложение OntoDictionary, которое предназначено для работы с научными математическими статьями и онтологиями, созданными в редакторе Protege. Приложение способно создавать словарь онтологии, разбивать его элементы на концепты и обрабатывать их в булевом поиске. Имеется функционал для выделения определённых именных групп из математических статей. Новизна заключается в создании и методе обработки именных групп, содержащих формулы. Формулы обрабатываются независимо от их типа. Построен отбор кандидатов в термины. По всему функционалу произведён ряд экспериментов с онтологией математического знания OntoMathPRO, которая также была разработана в Казанском федеральном университете.

Ключевые слова: *математическое знание, онтология, концепт, поисковый индекс, именная группа, кандидаты в термины.*

ВВЕДЕНИЕ

Как известно (см., например, [1]), в области искусственного интеллекта под онтологией понимается формальная система понятий и взаимосвязей между ними, описывающая определённую предметную область.

Онтологии можно классифицировать по уровню детализации, «природе» предметных действий, степени разработки и сопровождения. Развитие онтологий

актуально сейчас, так как они используются в системах автоматизированного сбора информации, управлении корпоративными информационными ресурсами, порталах и обучающих системах [2]. Особенно быстро онтологии совершенствуются в способах доступа к данным и методах автоматического построения вывода.

Отметим ряд языков проектирования онтологий. Например, LOOM, OKBC, OCML, Flogic, LBase – примеры традиционных языков. Ontolingua, CycL, SHOE – специальные языки спецификаций онтологий. XML, RDF, RDFS, OWL – это языки, основанные на веб-стандартах, их использование является наиболее современным и актуальным [3–5]. В настоящее время большинство мировых онтологических проектов использует RDF и OWL.

В настоящей статье рассмотрена задача создания удобного, современного программного инструмента для автоматизированного пополнения терминологического словаря предметной области и максимально полного анализа концептов онтологии.

5. ПОСТАНОВКА ЗАДАЧИ

Нашей целью является разработка информационной системы для автоматизированного пополнения терминологического словаря предметной области. Такая система должна иметь следующий функционал:

- конвертация содержимого онтологии в реляционное представление, сохраняющая изначальную древовидную структуру данных;
- реализация булевого поиска по концептам онтологии;
- разработка метода выделения из текста заданных NP-моделей;
- построение технологии отбора кандидатов в термины.

6. ОБЗОР ВЕДУЩИХ СОВРЕМЕННЫХ ИНСТРУМЕНТОВ ДЛЯ РАБОТЫ С ОНТОЛОГИЯМИ

Редактор Protege [6] разработан на языке Java в Стэндфордском университете и впервые представлен в 1987 году. С тех пор Protege не переставал развиваться: постоянно публиковались новые версии приложения, предназначенные для создания, просмотра и обновления онтологий. Строение онтологии является таким же, как и иерархия каталога. Имеется графический интерфейс. Основное

преимущество данного редактора – возможность адаптации приложения к моделям разных форматов (UML, OWL, XML, RDF, текстовый и ряд других). Такой функционал получил технологическую реализацию, благодаря фреймовой структуре представления знания [7] и ряду плагинов [8].

Система Ontolingua [9] так же, как редактор Protege, разработана в Стэндфордском университете и является самым ранним инструментом конструирования онтологий. Этот инструмент включает сервер, состоящий из архива онтологий многих предметных областей, и язык представления знаний (Ontolingua). Изменение онтологий реализовано через веб-приложение, где присутствует возможность визуализации концептов. Предусмотрено использование множественного наследования. Работа с форматами производится посредством сервера: предусмотрена конвертация онтологий из одного языка в другой.

Редактор OntoEdit [10] написан на языке Java и разработан в университете Karlsruhe. Это приложение функциональностью и структурой во многом похоже на Protege, но его коды закрыты. Программа имеет удобный интерфейс и подробную документацию. Кроме стандартных методов в интерфейс входят визуализация запросов и создание аксиом. Имеется возможность создания онтологий двумя способами: на основе логического вывода и с использованием методологии.

Редактор OilEd [11] создан в университете Манчестера и написан на языке OIL. В данный момент также поддерживается язык DAML+OIL. Редактор имеет функционал для вычисления логического противоречия классов и скрытых отношений концептов. Однако данное приложение не способно полноценно заменить аналоги, описанные выше, по следующим причинам:

- отсутствуют средства для слияния и изменения онтологий;
- нет возможности работы с большими данными;
- отсутствует контроль версий.

Следовательно, OilEd – своеобразный блокнот, в котором можно создавать, просматривать и исследовать онтологии, но не на профессиональном уровне.

OntoSaurus. Это приложение разработано в виде интернет-браузера, совмещает в себе как серверную, так и клиентскую части. На сервере хранятся базы

знаний LOOM [12]. Клиентская часть реализует красивый интерфейс с возможностью просмотра и изменения онтологий. Важно отметить, что основное предназначение программы – просмотр онтологий.

ODE (Ontological Design Environment) [13] – это инструмент для построения онтологий, отличительной чертой которого является связь с пользователем на уровне концептов, то есть данная программа предоставляет администратору таблицы (отношений, атрибутов, концептов), затем на основании введённой информации создаёт код в Loom и Ontolingua. Утверждается, что некоторым пользователям проще проектировать онтологии именно таким способом.

7. ОБЗОР ОСНОВНЫХ ИСПОЛЬЗУЕМЫХ МОДЕЛЕЙ

а. Модель реляционного представления онтологии

Реляционное представление информации имеет вид совокупности двумерных таблиц, состоящих из фиксированного количества столбцов и изменяемого количества строк. Для быстрого поиска по таблице применяются индексы. Кроме того, каждая таблица должна содержать первичный ключ, то есть поле, которое имеется во всех записях и при этом однозначно определяет их. Благодаря такой структуре, процедура поиска не зависит от иерархии информации, содержащейся в таблице.

В целях выполнения научно-исследовательской работы для хранения информации использована реляционная форма данных, так как описанная выше модель обладает подходящими свойствами, а именно:

- таблица легко представима на плоскости, то есть имеет двухмерную систему координат;
- обеспечена атомарность, то есть каждое поле является более неделимым;
- столбцы должны иметь разные названия;
- все ячейки одного столбца должны иметь одинаковые параметры, а именно, тип и длину;
- все строки в таблице различны;
- последовательность записей и полей может быть любой.

Для повсеместного эксплуатирования реляционных таблиц определены нормальные формы, то есть критерии, которым должна отвечать любая из таких

таблиц. Нормализация – это процесс приведения к этим обозначенным стандартам. Иначе говоря, он сводится к упрощению схемы базы данных. В нашем проекте все три таблицы были преобразованы к 1-ой и 2-ой нормальным формам, которые соответственно гласят:

- любое поле неделимо и не имеет повторяющихся групп;
- таблица уже приведена к 1-ой нормальной форме, и каждое её поле зависит от первичного ключа.

В приложении OntoDictionary применена реляционная база данных mysql, которая была создана с помощью инструмента MysSQL Workbench версии 6.3. Архитектура базы состоит из трёх таблиц (Concepts, Dictionary, ModelsNP) и изображена на рис. 1.

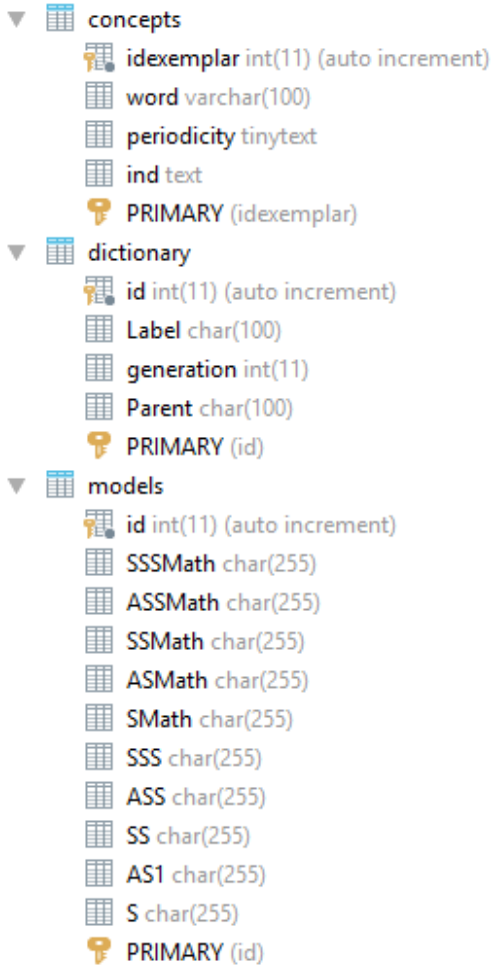

Рисунок 1. Архитектура базы данных

Таблица concepts содержит поля:

Idexemplar – тип целочисленный, авто инкремент для таблицы, первичный ключ;

Word – символьная строка, размером не более 100 символов;

Periodicity – строковая переменная, размером не более 255 байт;

Ind – текстовая переменная.

Таблица dictionary содержит поля:

Id – тип целочисленный, авто инкремент для таблицы, первичный ключ;

Label – символьная строка, размером не более 100 символов;

Generation – целочисленная переменная;

Parent – символьная строка, размером не более 100 символов.

Таблица Models содержит поля:

Id – тип целочисленный, авто инкремент для таблицы, первичный ключ;

SSSMath – символьная строка, размером не более 255 символов, которая предназначена для хранения моделей вида SS_2S_2Math;

ASSMath – символьная строка, размером не более 255 символов, которая предназначена для хранения моделей вида ASS_2Math;

SSMath – символьная строка, размером не более 255 символов, которая предназначена для хранения моделей вида SS_2Math;

ASMath – символьная строка, размером не более 255 символов, которая предназначена для хранения моделей вида $ASMath$;

SMath – символьная строка, размером не более 255 символов, которая предназначена для хранения моделей вида $SMath$;

SSS – символьная строка, размером не более 255 символов, которая предназначена для хранения моделей вида SS_2S_2;

ASS – символьная строка, размером не более 255 символов, которая предназначена для хранения моделей вида ASS_2;

SS – символьная строка, размером не более 255 символов, которая

предназначена для хранения моделей вида SS_2;

AS – символьная строка, размером не более 255 символов, которая предназначена для хранения моделей вида AS;

S – символьная строка, размером не более 255 символов, которая предназначена для хранения моделей вида S.

В разработанной базе данных можно совершать стандартные для реляционной модели операции, а именно: выделить (SELECT), вставить (INSERT), обновить (UPDATE), удалить (DELETE). Первая из них отбирает данные, с которыми будут производиться дальнейшие действия, а остальные три изменяют состояние базы данных. Пример:

- команда "SELECT * FROM dictionary" покажет всю информацию из таблицы dictionary;
- команда "INSERT INTO dictionary (Label, generation, Parent) VALUES ('" + label + "','" + generation + "','" + parent + "')" вставит в таблицу dictionary в поля Label, Generation, Parent соответственно значения переменных label, generation, parent;
- команда "UPDATE models SET SSSMath = ' " + NP + "' WHERE id = '" + j + "'" обновит значение поля SSSMath в таблице models, где поле id будет равно j. В результате, ячейка таблицы примет значение переменой NP, указанной в запросе.

b. Модель информационного поиска

Под информационным поиском мы понимаем процесс отбора информации из неструктурированного хранилища. Отбор осуществляется с помощью параметров, определяемых пользователем в зависимости от ожидаемого результата. Свободная структура хранилища означает отсутствие чёткой иерархии, фильтрации переменных. Стоит отметить, что сами параметры из хранилища всё же должны обладать необходимым строением, например, относиться к известному типу данных.

Существует несколько моделей поиска (булева, векторная, с использованием вероятностей и другие), однако на сегодняшний день среди них нет доминирующей, поэтому модель подбирается индивидуально под конкретную задачу.

В приложении OntoDictionary применена булева модель поиска, так как в данном случае она даёт точный результат. Модель считается наиболее простой,

при этом для описываемого инструмента она является достаточно эффективной. Она была изобретена в середине двадцатого века и до сих пор используется в крупных информационных службах.

Определение этой модели подразумевает возможность осуществления запросов, состоящих из совокупности слов и простейших логических операций, таких, как “И”, “ИЛИ”, “НЕ”. Эти операции необходимы для более детального отбора информации. Реализована модель посредством наличия поискового индекса по концептам. Для удобства применён интерфейс с возможностью выбора одного из четырёх вариантов поиска:

- первый вариант – поиск элементов, содержащих концепт из запроса пользователя;
- второй вариант – поиск элементов по сочетаниям из двух концептов, заданных в запросе;
- третий вариант – поиск элементов, имеющих хотя бы один из двух концептов, заданных пользователем;
- четвертый вариант – поиск всех элементов, которые не содержат концепт, заданный в запросе.

В нашем исследовании описанная выше система поиска построена на данных, взятых из словаря. Любой термин дробится на более мелкие части, концепты. В результате операции поиска пользователю предоставляется список, где каждый термин отвечает набору концептов, заданных в запросе.

Отметим, что понятия, хранящиеся в словаре, являются уникальными, поэтому вычислять частоту использования терминов не имеет смысла. А частота употребления концептов вычисляется программой и сохраняется в специально отведённом поле базы данных.

c. Семантические модели

Именная группа (Noun Phrase) является либо словосочетанием, в котором определено главное слово, либо одним именем существительным без зависимых слов. Основное слово называют ядром. В случае наличия зависимых слов оно определяет их характеристику. В текстах такие группы используются глаголами в роли объектов.

Чтобы вычислить NP (Noun Phrase), необходимо попытаться разрешить референциальные ссылки (местоимения). Пример:

-Диссертация содержит новый результат /Она содержит его.

В данном примере местоименные ссылки во второй фразе разрешаются на основе данных из первой фразы, т. к. «диссертация» и «новый результат» – это именные группы, удовлетворяющие грамматическим ограничениям. Видов NP очень много, так как размер словосочетаний в русском языке не ограничен. Как следствие, в нашей работе типы NP были отобраны самостоятельно, исходя из встречаемости в текстах выбранной тематики. Отметим, что OntoDictionary сконцентрирован на обработке математических научных статей, поэтому рассматриваются варианты, где в составе именной группы могут встречаться арифметические символы и формулы. Как следствие, нами было принято решение о разработке ранее не существующих NP-моделей, включающих формулы. Их проектирование, а также внедрение в информационную систему характеризуют научную новизну настоящей работы.

Использовались следующие обозначения:

S – имя существительное;

S_2 – имя существительное в родительном падеже;

A – прилагательное;

Math – формула.

С помощью этих обозначений нами были выделены именные группы:

AS – словосочетание из прилагательного и существительного, расположенных в соответствующем порядке;

SS_2 – словосочетание из двух существительных, второе из которых находится в родительном падеже;

ASS_2 – словосочетание из прилагательного и двух существительных; необходимо, чтобы второе существительное было в родительном падеже;

SS_2S_2 – словосочетание из трёх существительных, расположенных в соответствующем порядке; два последних слова должны находиться в родительном падеже;

S – любое существительное без зависимых слов;

$ASMath$ – словосочетание из согласованных слов: прилагательного, существительного и формулы, расположенных в соответствующем порядке;

SS_2Math – словосочетание из двух существительных и формулы, расположенных в указанном порядке;

ASS_2Math – словосочетание из прилагательного, двух существительных и формулы, представленных в указанном порядке;

SS_2S_2Math – словосочетание из четырёх слов: трёх последовательно расположенных существительных и формулы; два последних существительных должны быть в родительном падеже;

$SMath$ – словосочетание из существительного и формулы, представленных в соответствующем порядке.

Для определения именной группы и её характеристик был применен свободно распространяемый морфологический анализатор mystem, разработанный компанией Яндекс. Данный анализатор был выбран, так как он прост в использовании и обладает необходимым функционалом. Для работы с ним было применено несколько команд:

“*-n*” – применяется для вычисления нормальной формы слов, то есть вывода входного слова в единственном числе, именительном падеже, настоящем времени и так далее (в зависимости от части); если нормальную форму нельзя однозначно определить, анализатор предлагает несколько её вариантов;

“*-nig*” – используется для определения части речи и его семантических характеристик; в случае неоднозначности также выводится несколько вариантов.

8. ПРОГРАММНЫЕ ИНСТРУМЕНТЫ

Нами было создано веб-приложение, в котором объединены три разработанных инструмента. Для каждого из них имеется удобный интерфейс для взаимодействия с пользователем.

a. Инструмент конвертирования онтологии в реляционное представление

Задача этого инструмента состоит в том, чтобы обработать выбранную пользователем онтологию и сконвертировать её в реляционное представление. Интерфейс этого инструмента представлен на рис. 2. Программная реализация инструмента состоит из нескольких функций. Первая из них – TransformationFromOwlToTxt типа void, которая принимает на вход онтологию в качестве параметра. Работа функции осуществляется посредством библиотек apache.log4j, semanticweb.owlapi и semanticweb.elk.owlapi.ElkReasonerFactory.

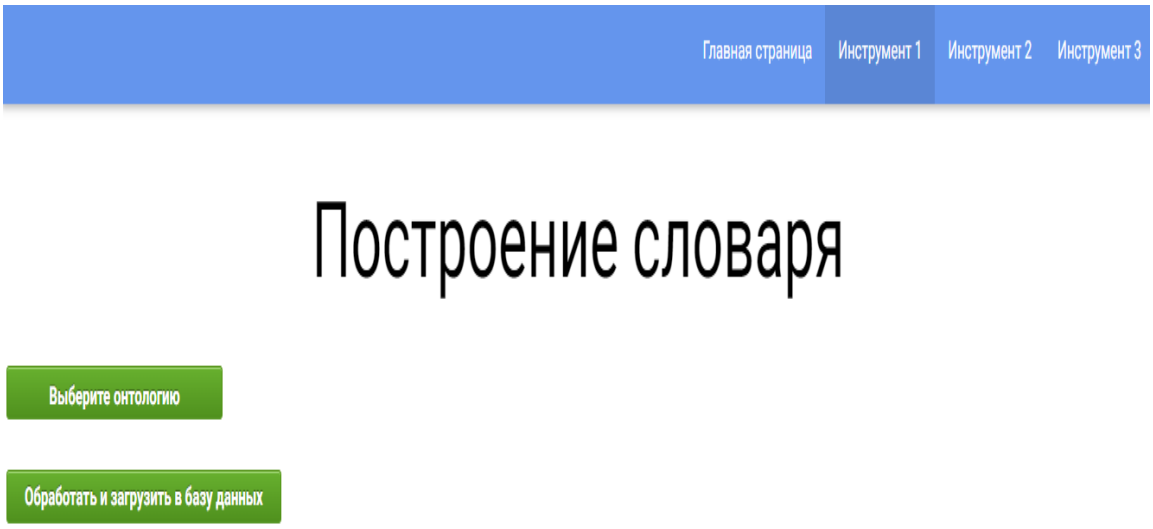

Рисунок 2. Интерфейс инструмента для построения словаря

Owlapi используется для парсера файлов такого формата. Парсер работает с использованием аргумента, reasoner. Поскольку в экспериментах на вход принималась усеченная онтология OntoMathPro, имеющая достаточно большой объём, то был использован специальный reasoner для работы с большими данными – ElkReasonerFactory.

Функция TransformationFromOwlToTxt реализует следующий алгоритм:

1. на основе входного файла создаётся новый объект класса SimpleHierarchy;

2. с использованием System.setOut указывается, что будущий выходной поток будет перенаправляется с консоли в указанный текстовый файл;

3. создаётся экземпляр ElkReasonerFactory;

4. создаётся экземпляр класса OWLClass, которому поочередно присваиваются значения классов онтологии;

5. у экземпляра класса SimpleHierarchy вызывается метод printHierarchy, принимающий в качестве аргументов экземпляр ElkReasonerFactory и экземпляр OWLClass;

6. в результате вызова printHierarchy печатается иерархия классов онтологии.

Результатом работы данной функции является файл input.txt с иерархией классов. Затем вызывается функция getStringSpaceCount для определения поколения каждого элемента иерархии. Алгоритм функции getStringSpaceCount:

1. создаётся пустой целочисленный список;

2. файл input.txt читается построчно;

3. в цикле, в каждой строке подсчитывается количество пробелов перед первым словом;

4. число, полученное на третьем шаге, добавляется в список.

Итогом работы данной функции является целочисленный список, числа в котором служат номерами поколений. Затем вызывается функция parents, которая определяет имя родительского элемента. Для самого верхнего элемента иерархии она укажет, что нет родителя. Функция принимает на вход два списка: лейблов (названий элементов) и поколений, а на выходе выдаёт список родительских элементов.

Последняя функция инструмента, writeDictionaryToDb, записывает все три списка (Labels, Generations, Parents) в таблицу Dictionary базы данных.

Алгоритм функции writeDictionaryToDb:

1. создаётся соединение с базой данных;

2. создаётся цикл for;

3. в цикле вычисляются переменные из трёх списков и вставляются в базу данных;

4. соединение с базой данных закрывается.

Как итог, имеем заполненную таблицу в базе данных. Затем данные через контроллер поступают из базы данных в соответствующую модель, Dictionary. Модель отправляет данные в представление. В результате пользователю выдается обновлённая таблица.

b. Инструмент поискового индекса по концептам онтологии

Данный инструмент предназначен для упрощения операции поиска элементов словаря. Интерфейс инструмента представлен на рис. 3. Используется механизм разбиения элементов на более мелкие части, концепты. Элементом словаря могут быть словосочетание, устойчивое выражение, условное обозначение или просто одиночное слово (например, термин). Концепт – это слово, семантическая единица языка. Таким образом, любой элемент построенного словаря предстаёт как совокупность логически связанных концептов.

За логическую связность в большинстве случаев отвечает форма слова. В данной задаче было принято решение абстрагироваться от всех возможных форм слов, то есть привести все концепты словаря к канонической форме. Такое решение позволило реализовать поиск, независимый от окончаний слов.

Приложение принимает на входе обработанный морфологическим анализатором файл, который содержит все концепты словаря. Таким образом, как промежуточный результат работы инструмента обработки OWL-файла, был создан файл с иерархией классов. Этот файл необходимо обработать морфологическим анализатором MyStem комадой “-n”.

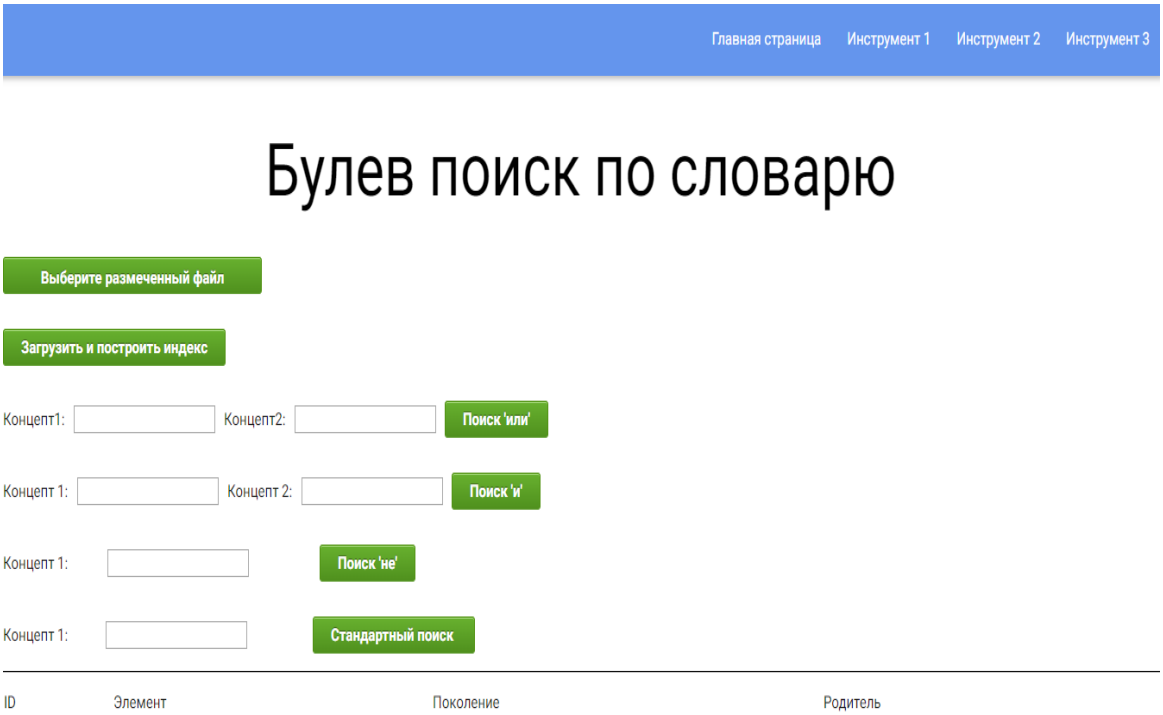

Рисунок 3. Интерфейс инструмента для булевого поиска по словарю

После того, как программа получает на вход нужный файл, она начинает обрабатывать его следующим образом:

1. Поскольку MyStem возвращает файл в формате, изображённом на рис. 4, необходимо очистить его от «мусора». Необходимо извлечь из файла только слова, то есть каждую вторую строчку. Для этого создаётся список clearList, и в функции afterSteamList он заполняется;
2. С помощью HashMap строится frequencyList, необходимый для вычисления частоты каждого концепта;
3. Создаётся и заполняется nonduplic – список без дубликатов;
4. Создаётся список index, где будут храниться индексы для каждого слова из списка nonduplic;

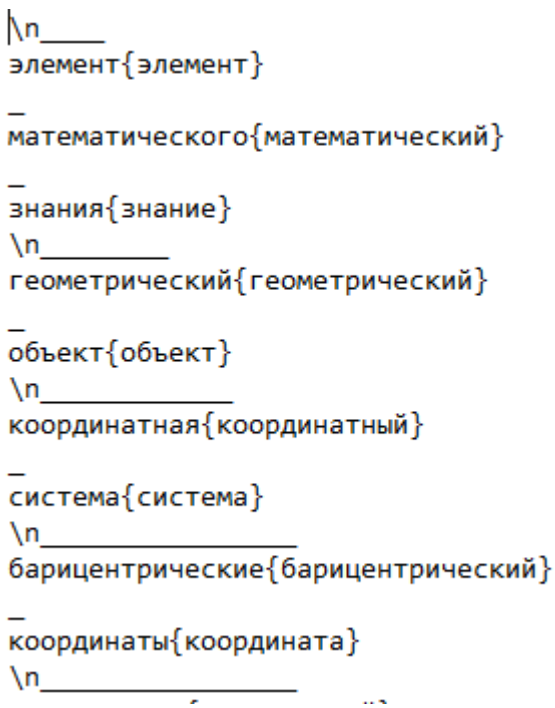

Рисунок 4. Структура файла после морфологического анализатора

5. Заполняется список index, в функции indexList, принимающей на вход в качестве аргументов два списка: index и labels – список названий элементов из словаря. Заполняется index в цикле, где он сравнивается с элементами списка nonduplic:

5.1. создаётся строка индексов ind="" для каждого слова;

5.2. находится индекс первого вхождения символа "{" в подстроке;

5.3. с помощью substring выбирается имя атрибута из подстроки, то есть строка считывается с нулевого элемента по номер элемента, найденный на шаге 5.2;

5.4. производится обход списка Labels; в цикле проверяется, содержит ли строка labels имя атрибута, полученного на шаге 5.3; если строка включает в себя искомое слово, то включается номер этой строки в строку индексов;

6. Добавляется строка индексов ind для определенного слова в общий список index;

7. От трёх переменных (списков frequencyList, nonduplic, index) вызывается функция записи концептов в таблицу concepts базе данных; в результате на данном этапе имеется полностью заполнения таблица для поиска элементов по индексу.

Как видно из интерфейса на рис. 3, имеется 4 вида поиска элементов по вводу концептов: стандартный, поиск «и», поиск «или», поиск «не».

Алгоритм стандартного поиска:

1. принимается концепт, введённый пользователем;

2. производится поиск этого концепта по таблице concepts;

3. в случае нахождения искомого концепта считывается значение его поля ind, то есть список id элементов таблицы Dictionary, которые содержат данный концепт;

4. из таблицы Dictionary выводятся строки с выбранными на шаге 3 значениями поля id;

5. если искомый концепт не находится, возвращается пустая таблица.

Алгоритм поиска «и»:

1. принимаются два концепта, введённые пользователем;

2. для каждого из этих концептов реализуется поиск соответствующего значения по таблице concepts;

3. в случае нахождения обоих концептов считываются их значения с полей ind; эти значения представляют набор целых чисел, разделённых пробелами; строятся два множества чисел, соответствующие значениям полей ind для каждого концепта; затем находится пересечение этих множеств, по которому строится запрос в таблицу Dictionary;

4. из таблицы Dictionary выводятся строки, значения id которых находятся в пересечении множеств, найденным на шаге 3;

5. если один из концептов не найден, то возвращается пустая таблица.

Алгоритм поиска «или»:

1. принимаются два концепта, введённые пользователем;.

2. для каждого из этих концептов реализуется поиск соответствующего значения по таблице concepts;

3. в случае нахождения обоих концептов считываются их значения с полей ind; эти значения представляют набор целых чисел, разделённых пробелами; строятся два множества чисел, соответствующие значениям полей ind для каждого концепта; затем находится объединение этих множеств, по которому строится запрос в таблицу Dictionary;

4. из таблицы Dictionary выводятся строки, значения id которых находятся в объединении множеств, найденных на шаге 3;.

5. если один из концептов не найден, то программа будет действовать, как при стандартном поиске, где результат ищется лишь по одному концепту.

Алгоритм поиска «не»:

1. принимается концепт, введённый пользователем;

2. производится поиск этого концепта по таблице concepts;

3. в случае нахождения искомого концепта считывается значение его поля ind, то есть список id элементов таблицы Dictionary, которые содержат данный концепт;

4. из таблицы Dictionary выводятся все строки, кроме тех, которые были выбраны на шаге 3;

5. если искомый концепт не находится, возвращается всё содержимое таблицы Dictionary, без исключений.

c. Инструмент извлечения терминологии из математических статей

Данный инструмент предназначен для работы с кандидатами в термины. Функционал представляет корректную обработку данных с трёх входных файлов и поиск по полученным результатам этой обработки. Интерфейс инструмента представлен на рис. 5.

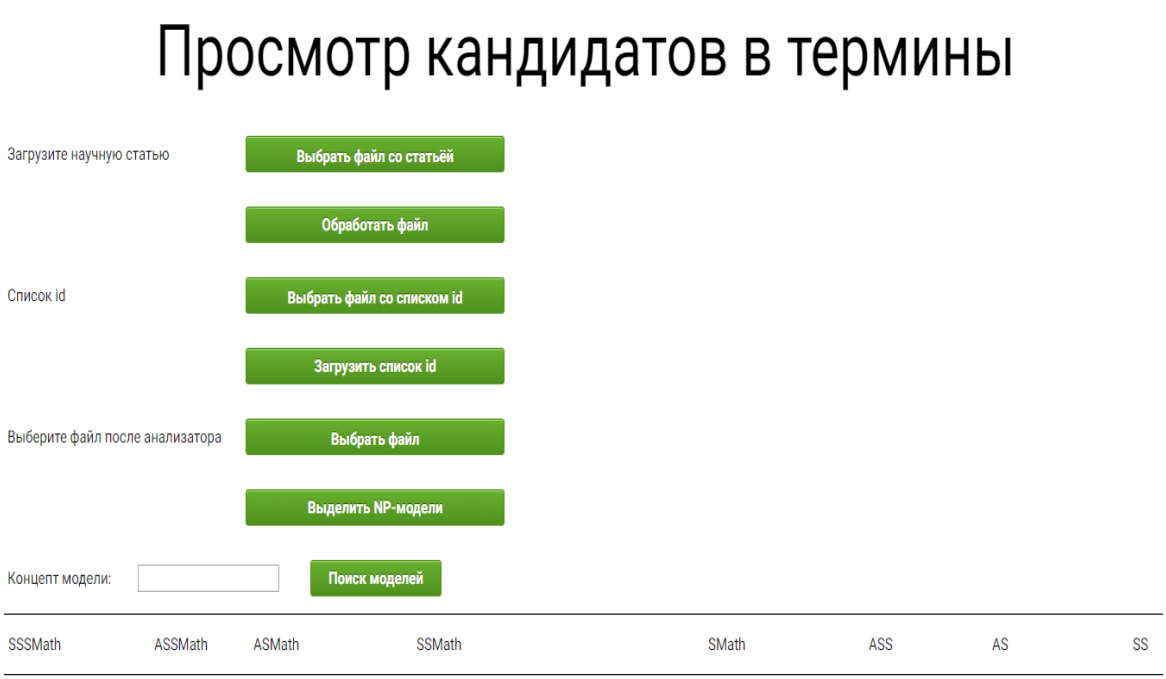

Рисунок 5. Интерфейс инструмента для просмотра и фильтрации кандидатов в термины

Для начала работы необходимо выбрать файл в формате xml и нажать кнопку «Обработать файл». После проверки на корректность файла запустится следующий алгоритм:

1. создаётся файл text.txt, куда будет записан результат работы функции;

2. с помощью библиотеки org.w3c.dom создаются экземпляры её классов Document, NodeList и Element; структура данных Document необходима для представления входного файла в нужном для библиотеки виде;

3. в экземпляр класса NodeList, названный nodeList, записывается всё содержимое файла, находящееся в теге “p”; отметим, что входному файлу не требуется определённая XML-разметка: программа универсальна, так как поддерживает любую структуру документа формата XML;

4. создаётся цикл в зависимости от размера nodeList;

5. в цикле из nodeList создаются экземпляры класса Element, которые построчно записываются в выходной файл;

6. закрывается соединение с записывающим файлом;

7. обновляется страница приложения, и появляется надпись, что файл успешно загружен.

Как итог работы функции, имеем текстовый файл с наличием всего текста из документа формата XML. Данный файл требуется обработать морфологическим анализатором MyStem, командой “-nig”. Затем пользователю следует выбрать заранее подготовленный список формул и их id и нажить кнопку «Загрузить список id». Этот файл будет загружен в корень и будет использован в функции, выделяющей NP-модели.

Следующая функция getNPResult принимает в качестве входного параметра файл «text.txt», обновлённый морфологическим анализатором. Для отсутствия путаницы теперь файл будет называться «Newtext.txt». Чтобы запустить функцию, необходимо выбрать файл и нажать кнопку «Выделить NP-модели».

Алгоритм функции getNPResult следующий:

1. обрабатывается загруженный файл со списком id для формул, то есть запускается цикл while:

1.1 пока не наступит конец файла, считывается строка;

1.2 если строка содержит подстроку «text=», то она добавляется в созданный список строк;

1.3 на выходе из цикла получается список lines, состоящий исключительно из нужных строк.

2. для каждой колонки таблицы models создаётся свой список, который будет заполняться в ходе реализации программы;

3. обрабатывается файл «Newtext.txt»:

3.1 вычисляется количество строк в файле;

3.2 строится массив, заполняющийся строками из файла;

4 создаётся цикл, где происходит работа с элементами массива (строками):

4.1 если строка содержит подстроку «S,», обозначающую, что в данной строке находится существительное, то делаются следующие шаги,

иначе программа переходит к следующей строке массива;

4.2 если следующая строка содержит подстроку «MathID», обозначающую, что в данной строке находится формула, то выполняется шаг 4.3, иначе сразу реализуется шаг 4.4;

4.3 из списка lines выбирается строка, из которой копируется формула и вставляется в элемент массива вместо её id;

4.4 на шаге 4.1 было выявлено, что текущий элемент является существительным; тогда проверяется, является ли предыдущий элемент существительным и находится ли текущий элемент в родительном падеже; если проверка показала, что это так, проверяется, стоит ли согласованное прилагательное перед ними, то есть существует ли такой объект массива и содержит ли он подстроку «A,» (обозначение прилагательного), а также совпадают ли его семантические характеристики (род, число, падеж) с семантическими характеристиками существительного, которое идёт следом (относительно текущего элемента, это предыдущий элемент); если всё совпало, то элементы массива добавляются, в зависимости от наличия формулы, соответственно в список ASSMath либо ASS;

4.5 аналогичным образом с помощью сравнений выделяются остальные NP-группы; выделим два семантических правила, которые использовались при программировании алгоритма:

- если рядом находятся два существительных, то второе из них обязательно должно быть в родительном падеже; прочие семантические характеристики у них не сравниваются;
- если рядом находятся прилагательное и существительное, то всегда проверяется, чтобы они были согласованными; для этого необходимо, чтобы у них совпали род, число и падеж;

5. полученные списки NP вставляются в таблицу models базы данных; для удобства просмотра было принято решение построения такой архитектуры, где строка таблица представляла бы 10 разных элементов, а не 1 элемент с десятью свойствами, как принято в классическом представлении; для записи в такую таблицу используется следующий алгоритм:

5.1. из 10-ти списков NP вычисляется размер самого большого;

5.2. в зависимости от этого размера в таблице базы данных создаётся число пустых строк;

5.3. при записи списков NP значения пустых строк обновляются.

6. полученные значения из таблицы показываются пользователю.

Также имеется функционал для отбора из построенных NP-моделей кандидатов в термины. Для использования метода отбора необходимо ввести концепт в поле для ввода и нажать кнопку «Поиск кандидатов».

Алгоритм отбора кандидатов в термины:

1. принимается введённый пользователем концепт модели;

2. производится поиск этого концепта по всем столбца таблицы models;

3. если какое-то поле содержит искомый концепт, то строка с этим полем считывается и добавляется в список result;

4. в списке result проверяются все поля каждой строки; если поле не содержит нужного концепта, то для текущего запроса его значение становится пробелом;

5. в результате запроса изображается отфильтрованная таблица models;

6. результат запроса записывается в файл «Результат поиска.txt»;

7. если искомый концепт не находится, возвращается пустая таблица.

9. ПРОВЕДЁННЫЕ ЭКСПЕРИМЕНТЫ С OntoMathPro

Эксперимент с применением инструмента конвертирования онтологии в реляционное представление. Этот эксперимент проводился с усечённой версией специализированной онтологии математической области OntoMathPro. Результат обработки файла представлен на рисунках 6–9, на которых, соответственно, изображены фрагмент онтологии, часть выходного текстового файла, фрагмент таблицы, заполненной в результате работы инструмента, и фрагмент таблицы, отображенной в приложении.

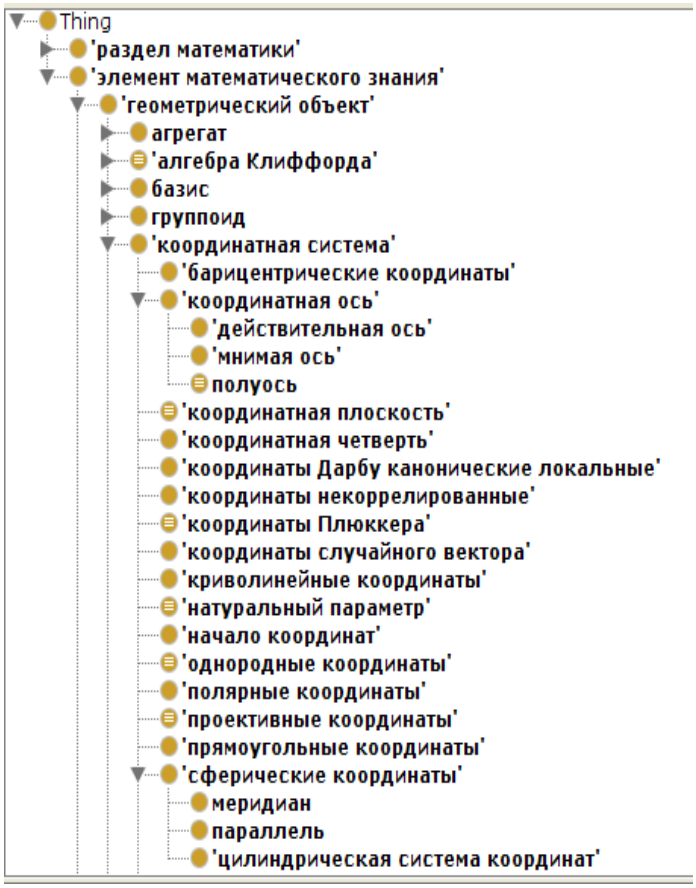

Рисунок 6. Фрагмент, использованной в ходе экспериментов, онтологии

```
элемент математического знания
    геометрический объект
        координатная система
            барицентрические координаты
            проективные координаты
            полярные координаты
            координатная плоскость
            координатная четверть
            координаты Плюккера
            координаты Дарбу канонические локальные
            сферические координаты
                меридиан
                цилиндрическая система координат
                параллель
            координаты случайного вектора
            координаты некоррелированные
            координатная ось
                полуось
                действительная ось
                мнимая ось
            натуральный параметр
            цилиндрические координаты
            криволинейные координаты
            однородные координаты
            начало координат
            декартова система координат
```

Рисунок 7. Фрагмент выходного файла

id	Label	generation	Parent
1	элемент математического знания	1	no
2	геометрический объект	2	элемент математического знания
3	координатная система	3	геометрический объект
4	барицентрические координаты	4	координатная система
5	проективные координаты	4	координатная система
6	полярные координаты	4	координатная система
7	координатная плоскость	4	координатная система
8	координатная четверть	4	координатная система
9	координаты Плюккера	4	координатная система
10	координаты Дарбу канонические ло…	4	координатная система
11	сферические координаты	4	координатная система
12	меридиан	5	сферические координаты
13	цилиндрическая система координат	5	сферические координаты
14	параллель	5	координатная система
15	координаты случайного вектора	4	сферические координаты
16	координаты некоррелированные	4	сферические координаты
17	координатная ось	4	координатная система
18	полуось	5	координатная ось
19	действительная ось	5	координатная ось
20	мнимая ось	5	координатная система
21	натуральный параметр	4	координатная ось
22	цилиндрические координаты	4	координатная ось
23	криволинейные координаты	4	координатная ось
24	однородные координаты	4	координатная ось
25	начало координат	4	координатная ось

Рисунок 8. Заполненная в ходе экспериментов таблица Dictionary

ID	Элемент	Поколение	Родитель
1	элемент математического знания	1	no
2	геометрический объект	2	элемент математического знания
3	координатная система	3	геометрический объект
4	барицентрические координаты	4	координатная система
5	проективные координаты	4	координатная система
6	полярные координаты	4	координатная система
7	координатная плоскость	4	координатная система
8	координатная четверть	4	координатная система
9	координаты Плюккера	4	координатная система
10	координаты Дарбу канонические локальные	4	координатная система

Рисунок 9. Отображение заполненной таблицы Dictionary

На всех четырёх рисунках показаны одни и те же данные, и по ним сделан вывод, что программа работает корректно и успешно конвертирует данные в реляционное представление.

Эксперимент с применением инструмента для поискового индекса по концептам онтологии. Этот эксперимент проводился со словарём, построенным в предыдущем эксперименте. Часть результата обработки этого файла представлена на рис. 10, где видно, что в базу данных была записана верная информация. Чтобы удостовериться в правильности построения индекса, разберём конкретный пример – слово «координатная», находящееся на шестой строчке рис. 10. Видно, что оно имеет частоту, равную 4, и используется в элементах словаря с id, равным 3, 7, 8, 17. Если посмотреть на рис. 8, то можно удостовериться, что это верная информация.

idexemplar	word	periodicity	ind
1	элемент{элемент}	7	1 30 579 762 1922 1973 2349
2	математического{математический}	12	1 762 992 1168 1189 1233 1245 1413 1574 1734 2392 …
3	знания{знание}	1	1
4	геометрический{геометрический}	2	2 1379
5	объект{объект}	2	2 1189
6	координатная{координатный}	4	3 7 8 17
7	система{система}	13	3 13 26 220 323 325 336 338 340 805 806 807 808
8	барицентрические{барицентрический}	1	4
9	координаты{координата}	11	4 5 6 9 10 11 15 16 22 23 24
10	проективные{проективный}	1	5
11	координаты{координата}	1	6
12	полярные{полярный}	16	7 1055 1070 1078 1118 2042 2043 2044 2097 2219 223…
13	координаты{координата}	1	8
14	координатная{координатный}	1	9
15	плоскость{плоскость}	5	10 924 1205 1664 2014
16	координатная{координатный}	1	10
17	четверть{четверть}	1	10
18	координаты{координата}	1	11
19	Плюккера{плюккер?}	1	12
20	координаты{координата}	1	13
21	Дарбу{дарба?|дарб?}	18	3 4 5 6 7 8 9 10 11 13 15 16 17 22 23 24 25 26
22	канонические{канонический}	4	14 200 222 640
23	локальные{локальный}	1	15
24	сферические{сферический}	1	15
25	координаты{координата}	1	16

Рисунок 10. Заполненная в ходе экспериментов таблица Concepts

Также были проведены успешные эксперименты с четырьмя типами поиска. По результатам всех экспериментов сделан вывод, что инструмент работает корректно и реализует алгоритм булевого поиска на основе концептов.

Эксперимент с применением инструмента для извлечения терминологии из математических статей. Этот эксперимент проводился с входными файлами, фрагменты которых представлены на рисунках 11–13. На рис. 11 показано, как встречаются id формул среди текста. На рис. 12 видно, что в начале каждой строки пишется id, а само значение формулы находится в поле text тега Math. На рис. 13 приведён пример синтаксиса файла после обработки MyStem. Результат обработки файлов представлен на рис. 14, где изображён фрагмент заполненной таблицы базы данных. На рис. 15 показано отображение заполненной таблицы на страницу инструмента.

<?xml version="1.0" encoding="UTF-8"?><?latexml searchpaths="/opt/mocassin/patched-tex,/home/linglab/projects/thirdparty/a
d</ERROR> <para xml:id="p2"> 20052 (513)<ERROR class="undefined">\nomer</ERROR><!-- %Remove in Eng. %\nomer{Vo
есны свойства периферического спектра линейных операторов.Периферический спектр оператора MathIDA, некоторая степень котор
нтированный путь MathIDBI.Число звеньев пути называется длиной этого пути.Однозвенный путь из MathIDBJ в MathIDCA (петля)
*** -->общему делителю разностейMathIDDD.Известна и другая характеристика индекса импримитивности ([2], с. 131):MathIDDE e
OR class="undefined"><XMTok role="UNKNOWN" font="italic">\ov</XMTok></ERROR>.Тогда MathIDFA будет единственным с точностью
яет уравнению MathIDFJ<ERROR class="undefined"><XMTok role="UNKNOWN" font="italic">\ov</XMTok></ERROR>.Утверждение следует
твMathIDHF,то MathIDHG. </para> <para xml:id="p12"> Оценим сверху норму однорангового возмущения MathIDHH. Пус
го возмущения MathIDJE имеем оценку <equation xml:id="S0.Ex3"> MathIDJF </equation> При MathIDJG, пол
./../mocassin/patched-tex/ivm12.tex Line 200 **** --> <para xml:id="p15"> Аналогом неприводимой матрицы с неотрицат
естно ([1], с. 337), что свойство s) эквивалентно следующему свойству:существует такой скаляр MathIDBDI, что для всякого M
thIDBFH иMathIDBFI будут единственными с точностью до постоянногомножителя собственными векторами, отвечающими соответстве
belref="LABEL:4"/>). </para> <para xml:id="p20"> При фиксированном MathIDBHJ имеем <equation xml:id="S0
ara xml:id="p22"> В общем случае для потенциально компактного положительного оператора MathIDBJH,неприводимого над з
заданных насвязном хаусдорфовом компакте MathIDCAI. </para> </item> </itemize> </para> <para xml:id="

Рисунок 11. Фрагмент статьи формата xml, использованной в эксперименте

Также был проведен эксперимент по извлечению кандидатов в термины на основе концептов. Был использован концепт «Элемент». В итоге были извлечены все кандидаты в термины, содержащие этот концепты. Фрагмент результирующей таблицы представлен на рис. 16, где также видно, какому классу NP-моделей принадлежат отобранные кандидаты.

```
A   F2  <Math mode="inline" tex="A" xml:id="p6.m1" text="A"><XMath><XMTok role="UNKNOWN" font="italic">A</XMTok></XMath></Math>
B   F1  <Math mode="inline" tex="\{\lambda:|\lambda|=r(A)\}" xml:id="p6.m2" text="conditional-set@(lambda, absolute-value@(lambda) = r * A)">
C   F2  <Math mode="inline" tex="r(A)" xml:id="p6.m3" text="r * A"><XMath><XMApp><XMTok meaning="times" role="MULOP">x</XMTok><XMTok role="U
D   F2  <Math mode="inline" tex="A" xml:id="p6.m4" text="A"><XMath><XMTok role="UNKNOWN" font="italic">A</XMTok></XMath></Math>
E   F2  <Math mode="inline" tex="A" xml:id="p7.m1" text="A"><XMath><XMTok role="UNKNOWN" font="italic">A</XMTok></XMath></Math>
F   F2  <Math mode="inline" tex="n" xml:id="p7.m2" text="n"><XMath><XMTok role="UNKNOWN" font="italic">n</XMTok></XMath></Math>
G   F2  <Math mode="inline" tex="a_{{ij}}" xml:id="p7.m3" text="a _ (i * j)"><XMath><XMApp><XMTok role="SUBSCRIPTOP" scriptpos="post2"/><XMTo
H   F2  <Math mode="inline" tex="n" xml:id="p7.m4" text="n"><XMath><XMTok role="UNKNOWN" font="italic">n</XMTok></XMath></Math>
I   F2  <Math mode="inline" tex="p_{1},\ldots,p_{n}" xml:id="p7.m5" text="list@(p _ 1, ldots, p _ n)"><XMath><XMApp><XMTok meaning="list" rol
J   F2  <Math mode="inline" tex="a_{{ij}}" xml:id="p7.m6" text="a _ (i * j)"><XMath><XMApp><XMTok role="SUBSCRIPTOP" scriptpos="post2"/><XMTo
BA  F2  <Math mode="inline" tex="A" xml:id="p7.m7" text="A"><XMath><XMTok role="UNKNOWN" font="italic">A</XMTok></XMath></Math>
BB  F2  <Math mode="inline" tex="p_{i}" xml:id="p7.m8" text="p _ i"><XMath><XMApp><XMTok role="SUBSCRIPTOP" scriptpos="post2"/><XMTok role="U
```

Рисунок 12. Фрагмент списка id, использованного в эксперименте

```
Спектральные{спектральный=A=(вин,мн,полн,неод|им,мн,полн)}
свойства{свойство=S,сред,неод=(вин,мн|род,ед|им,мн)}
матриц{матрица=S,жен,неод=род,мн}
с{с=PR=|с=S,сокр=(пр,мн|пр,ед|вин,мн|вин,ед|дат,мн|дат,ед|род,мн|род,ед|твор,мн|твор,ед|им,мн|им,ед)}
вещественными{вещественный=A=твор,мн,полн}
неотрицательными{неотрицательный=A=твор,мн,полн}
элементами{элемент=S,муж,неод=твор,мн}
установленные{устанавливать=V,пе=(прош,вин,мн,прич,полн,сов,страд,неод|прош,им,мн,прич,полн,сов,страд)|установленный=A=(вин,мн,полн,неод|им,мн,полн)}
Перроном{перрон=S,муж,неод=твор,ед}
и{и=CONJ=|и=INTJ=|и=S,сокр=(пр,мн|пр,ед|вин,мн|вин,ед|дат,мн|дат,ед|род,мн|род,ед|твор,мн|твор,ед|им,мн|им,ед)|и=PART=}
фробениусом{фробениус?=S,фам,муж,од=твор,ед|фробениус?=S,имя,муж,од=твор,ед}
в{в=PR=|в=S,сокр=(пр,мн|пр,ед|вин,мн|вин,ед|дат,мн|дат,ед|род,мн|род,ед|твор,мн|твор,ед|им,мн|им,ед)}
начале{начало=S,сред,неод=пр,ед}
прошлого{прошлое=S,сред,неод=род,ед|прошлый=A=(вин,ед,полн,муж,од|род,ед,полн,муж|род,ед,полн,сред)}
столетия{столетие=S,сред,неод=(вин,мн|род,ед|им,мн)}
были{быть=V,нп=прош,мн,изъяв,несов|быль=S,жен,неод=(пр,ед|вин,мн|дат,ед|род,ед|им,мн)}
обнаружены{обнаруживать=V,пе=прош,мн,прич,кр,сов,страд}
сначала{сначала=ADV=}
у{у=PR=|у=INTJ=|у=S,сокр=(пр,мн|пр,ед|вин,мн|вин,ед|дат,мн|дат,ед|род,мн|род,ед|твор,мн|твор,ед|им,мн|им,ед)}
интегральных{интегральный=A=(пр,мн,полн|вин,мн,полн,од|род,мн,полн)}
операторов{оператор=S,муж,од=(вин,мн|род,мн)|оператор=S,муж,неод=род,мн}
а{а=CONJ=|а=PART=|а=INTJ=|а=S,сокр=(пр,мн|пр,ед|вин,мн|вин,ед|дат,мн|дат,ед|род,мн|род,ед|твор,мн|твор,ед|им,мн|им,ед)}
затем{затем=ADV=}
и{и=CONJ=|и=INTJ=|и=S,сокр=(пр,мн|пр,ед|вин,мн|вин,ед|дат,мн|дат,ед|род,мн|род,ед|твор,мн|твор,ед|им,мн|им,ед)|и=PART=}
у{у=PR=|у=INTJ=|у=S,сокр=(пр,мн|пр,ед|вин,мн|вин,ед|дат,мн|дат,ед|род,мн|род,ед|твор,мн|твор,ед|им,мн|им,ед)}
более{более=ADV=|много=ADV=срав}
общих{общий=A=(пр,мн,полн|вин,мн,полн,од|род,мн,полн)}
операторов{оператор=S,муж,од=(вин,мн|род,мн)|оператор=S,муж,неод=род,мн}
```

Рисунок 13. Фрагмент файла после обработки морфологическим анализатором

ASSMath
спектральный радиус оператор "conditional-set@(lambda, absolute-value@(lambda) = r * A)"
квадратный матрица порядок "conditional-set@(lambda, absolute-value@(lambda) = r * A)"
направлять граф матрица "list@(p _ 1, ldots, p _ n)"
однозвенный? путь из "p _ i"
периферический спектр матрица "A"
характеристический многочлен матрица "A"
общий делитель разность "Delta * lambda = lambda ^ n + a _ 1 * lambda ^ (n _ 1) + cdots + a _ t * lambda ^ (n _ t)"
направлять граф матрица "A"
положительный решение уравнение "A ^ * * v = r * A * v"
собственный вектор матрица "A ^ *"
направлять граф матрица "A"
квазивнутренний? точка решетка "B"
квазивнутренний? точка решетка "B"
это неравенство при "r * A = 1"
спектральный радиус оператор "A"
квазивнутренний? точка решетка "B"
единственный решение уравнение "x = A * x + f"

Рисунок 14. Столбец таблицы models, заполненной в эксперименте

SSSMath	ASSMath	ASMath	SSMath	SMath	ASS	AS	SS
произведение вектор в "C ^ n"	спектральный радиус оператор "conditional-set@(lambda, absolute-value@(lambda) = r * A)"	ненулевой элемент "a _ (i * j)"	вектор матрица "A"	плоскость "n"	собственный значение этот	периферический спектр	перрон и
и из равенство "A ^ * * v = r * A * v"	квадратный матрица порядок "conditional-set@(lambda, absolute-value@(lambda) = r * A)"	ориентированный путь "list@(overline@(p _ i * p _ i _ 1), ldots, overline@(p _ i _ l * p _ j))"	индекс импримитивность? "m"	точка "a _ (i * j)"	этот матрица и	некоторый степень	фробениус? в
кронекер? из равенство "absolute-value@(D ^ (k - 1) * x) = absolute-value@(x)"	направлять граф матрица "list@(p _ 1, ldots, p _ n)"	спектральный радиус "overline@(p _ i * p _ j)"	идеал решетка "B"	матрица "A"	другой характеристика индекс	спектральный радиус	начало прошлое

Рисунок 15. Отображение фрагмента заполненной таблицы models

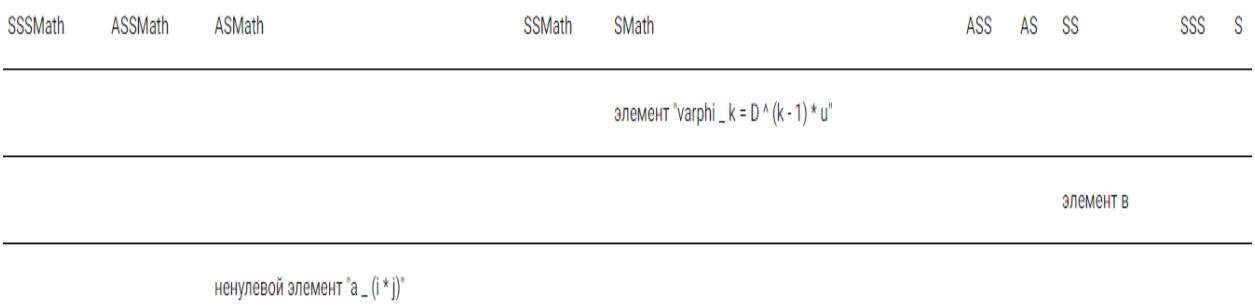

Рисунок 16. Извлечённые кандидаты для концепта «элемент»

По итогам экспериментов сделан вывод, что инструмент работает корректно: правильно строит NP-модели и извлекает кандидатов в термины.

ЗАКЛЮЧЕНИЕ

В статье представлен разработанный программный инструмент для работы с онтологиями, который может быть использован для автоматизированного пополнения терминологического словаря предметной области. Проведенные эксперименты показали корректность полученных результатов. В дальнейшем планируется усовершенствовать данный инструмент с помощью анализа семантических отношений между классами.

БЛАГОДАРНОСТИ

Работа выполнена за счет средств субсидии, выделенной Казанскому федеральному университету для выполнения государственного задания в сфере научной деятельности, проект 1.2368.2017/ПЧ.

СПИСОК ЛИТЕРАТУРЫ

1. Портал искусственного интеллекта. 2017. URL: http://www.aiportal. ru/articles/other/ontology.html

2. *Ле Хоай, Тузовский А.Ф.* Использование онтологии в электронных библиотеках // Изв. Томского политехнического университета. – 2012. – Т. 320, № 5. – С. 36–42.

3. Обзор языка онтологии OWL. 2017 URL: https://www.w3.org/TR/owl-features/ (Дата обращения: 20.12.2017)

4. Словарь RDF. Язык описания 1.1 URL схемы RDF. URL: https://www.w3. org/TR/rdf-schema/ (Дата обращения: 20.12.2017)

5. Структура описания ресурсов (RDF). 2017. URL: https://www.w3.org/RDF/ (Дата обращения: 20.12.2017)

6. *Musen M.* Domain Ontologies in Software Engineering: Use of Protege with the EON Architecture // Methods of Information in Medicine. – 1998. – P. 540–550.

7. *Chaudhri V., Farquhar A., Fikes R., Karp P., Rice J.* OKBC: A Programmatic Foundation for Knowledge Base Interoperability // Fifteenth National Conf. on Artificial Intelligence. AAAIPres/ The MIT Press, Madison, 1998. – P. 600–607.

8. *Noy N., Sintek M., Decker S., Crubezy M., Fergerson R., Musen M.* Creating Semantic Web Contents with Protege-2000 // IEEE Intelligent Systems, March/April, 2001 – P. 60–71.

9. *Farquhar A., Fikes R., Rice J.* The Ontolingua Server: A Tool for Ccollaborative Ontology Construction // Int. J. of Human-Computer Studies. 1997. – Vol. 46, No 6. – P. 707–728.

10. *Sure Y., Erdmann M., Angele J., Staab S., Studer R., Wenke D.* OntoEdit: Collaborative ontology development for the Semantic Web // In Proc. of the Int. Semantic Web Conference (ISWC 2002), Sardinia, Italia, June 2002.

11. *Bechhofer S., Horrocks I., Goble C., Stevens R.* OilEd: A Reasonable Ontology Editor for the Semantic Web // Joint German/Austrian Conf. on Artificial Intelligence (KI'01). Lecture Notes in Artificial Intelligence LNAI 2174, Springer-Verlag, Berlin, 2001. – P. 396–408.

12. *MacGregor R.* Inside the LOOM classifier // SIGART Bulletin. – 1991. – Vol. 3, No 2. – P. 70–76.

13. *Fernandez M, Gomez-Perez A., Pazos J.* A Building a Chemical Ontology Using Methodology and the Ontology Design Environment // IEEE Intelligent Systems, Jan./Feb., 1999. – P. 37–46.

SOME SOFTWARE INSTRUMENTS FOR THE AUTOMATED REPLENISHMENT OF THE TERMINOLOGICAL DICTIONARY OF THE SUBJECT OBLAST

[1]R. A. Rumyantsev, [1,2]O. A. Nevzorova

[1]Institute of Computational Mathematics and Information Technologies, Kazan (Volga Region) Federal University, Kazan, 35 Kremlyovskaya Street, 420008
[2]Institute of Applied Semiotics, Academy of Academy of Sciences of the Republic of Tatarstan, Kazan, Levo-Bulachnaya Street, 36A, 420111

Abstract

The article describes the application OntoDictionary, which is designed to work with scientific mathematical articles and ontologies created in Protege. The application is able to create an ontology dictionary, split its elements into concepts, and process them in Boolean search. There is a functional for the selection of certain nominal groups from mathematical articles. The novelty lies in the creation and method of processing nominal groups containing formulas. Formulas are processed regardless of their type. The selection of candidates for terms has been constructed. Throughout the functional, a number of experiments have been performed with the ontology of mathematical knowledge of OntoMathPRO, which was also developed at the Kazan Federal University.

Keywords*: mathematical knowledge, ontology, concept, search index, Noun Phrase, candidates in terms.*

REFERENCES

1. Portal of artificial intelligence. 2017 URL: http://www.aiportal.ru/ articles/other/ontology.html

2. *Le Howay, Tuzovsky A.F.* Using ontology in electronic libraries // Proc. of the Tomsk Polytechnic University. – 2012. – № 5. – S. 36–42.

3. OWL Web Ontology Language Overview. 2017 URL: https://www.w3.org/TR/owl-features/

4. RDF Vocabulary Description Language 1.1 RDF Schema URL: https://www.w3.org/TR/rdf-schema/

5. Resource Description Framework (RDF). 2017. URL: https://www.w3.org/RDF6.

6. *Musen M.* Domain Ontologies in Software Engineering: Use of Protege with the EON Architecture // Methods of Information in Medicine. – 1998. – P. 540–550.

7. *Chaudhri V., Farquhar A., Fikes R., Karp P., Rice J.* OKBC: A Programmatic Foundation for Knowledge Base Interoperability // Fifteenth National Conf. on Artificial Intelligence. AAAIPres/ The MIT Press, Madison, 1998. – P. 600–607.

8. *Noy N., Sintek M., Decker S., Crubezy M., Fergerson R., Musen M.* Creating Semantic Web Contents with Protege-2000 // IEEE Intelligent Systems, March/April, 2001 – P. 60–71.

9. *Farquhar A., Fikes R., Rice J.* The Ontolingua Server: A Tool for Ccollaborative Ontology Construction // Int. J. of Human-Computer Studies. 1997. – Vol. 46, No 6. – P. 707–728.

10. *Sure Y., Erdmann M., Angele J., Staab S., Studer R., Wenke D.* OntoEdit: Collaborative ontology development for the Semantic Web // In Proc. of the Int. Semantic Web Conference (ISWC 2002), Sardinia, Italia, June 2002.

11. *Bechhofer S., Horrocks I., Goble C., Stevens R.* OilEd: A Reasonable Ontology Editor for the Semantic Web // Joint German/Austrian Conf. on Artificial Intelligence (KI'01). Lecture Notes in Artificial Intelligence LNAI 2174, Springer-Verlag, Berlin, 2001. – P. 396–408.

12. *MacGregor R.* Inside the LOOM classifier // SIGART Bulletin. – 1991. – Vol. 3, No 2. – P. 70–76.

13. *Fernandez M, Gomez-Perez A., Pazos J.* A Building a Chemical Ontology Using Methodology and the Ontology Design Environment // IEEE Intelligent Systems, Jan./Feb., 1999. – P. 37–46.

СВЕДЕНИЯ ОБ АВТОРАХ

РУМЯНЦЕВ Роман Анатольевич – Магистрант Института вычислительной математики и информационных технологий Казанского (Приволжского) федерального университета, Республика Татарстан, г. Казань

Roman A. RUMYANTSEV – graduate student in the Institute of Computational Mathematics and Information Technologies of the Kazan (Volga Region) Federal University, Republic of Tatarstan, Kazan

email: romanrar93@gmail.com

НЕВЗОРОВА Ольга Авенировна – кандидат технических наук, доцент кафедры информационных систем Института вычислительной математики и информационных технологий Казанского (Приволжского) федерального университета, Республика Татарстан, г. Казань

Olga A. NEVZOROVA – Candidate of Technical Sciences, Associate Professor of the Information Systems Department of the Institute of Computational Mathematics and Information Technologies of the Kazan (Volga Region) Federal University, Republic of Tatarstan, Kazan

email: onevzoro@gmail.com

Материал поступил в редакцию 15 мая 2018 года