ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ

Отчет о прохождении производственной технологической (проектно-технологической) практической подготовки на тему: «Исследование возможностей Tomita-парсера для извлечения тональной информации из текстовых сообщений»

Петровой Марии Петровны
3 курс, гр. 3530903/80301

Направление подготовки: 09.03.03 Прикладная информатика.
Место прохождения практической подготовки: СПбПУ, ИКНТ, ВШИСиСТ г. Санкт-Петербург, ул. Обручевых, д. 1, лит. В.
Сроки практической подготовки: с 05.06.2021 по 03.07.2021.
Руководитель практической подготовки от ФГАОУ ВО «СПбПУ»: Пархоменко Владимир Андреевич, ассистент ВШИСиСТ.
Консультант практической подготовки от ФГАОУ ВО «СПбПУ»: Туральчук Константин Анатольевич, к. т. н., доцент ВШИСиСТ.
Оценка: _____

Руководитель практической подготовки
от ФГАОУ ВО «СПбПУ» В.А. Пархоменко

Консультант практической подготовки
от ФГАОУ ВО «СПбПУ» К.А. Туральчук

Обучающийся М.П. Петрова

Дата: 03.07.2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На данный момент информация хранится преимущественно в электронном виде. Обработка текстовых сообщений представляет собой большой интерес в различных сферах. Например, отзывы потребителей товаров или услуг в Интернете позволяют производителям улучшить качество предоставляемой продукции, выявить потребности потребителей, а потребителям сделать выбор в пользу того или иного товара или услуги. Анализ комментариев в социальных помогает отслеживать и удалять комментарии, содержащие в себе оскорбления, призывы к противоправным действиям. Анализ текстовых сообщений используется во многих сферах науки и бизнеса. Текстовые сообщения представляют собой текст на естественном языке, так людям привычно воспроизводить и воспринимать информацию, но для обрабатывать большие объемы данных, написанных таким способом, затруднительно.

Для извлечения структурированных данных из текстовых сообщений, написанных на естественном языке, существует большое количество инструментов (в особенности, для английского языка). Для русского языка существует инструмент от компании Яндекс - "Томита-парсер". Для обработки полученной структурированной информации используются методы компьютерной лингвистики. Одной из задач компьютерной лингвистики является анализ тональности текста. Тональность текста - это эмоциональное отношение автора текста к какому-то объекту. Тональность часто определяется как "позитивная"или "негативная также есть другие оценки. Методами определения тональности могут быть основаны на использовании тональных словарей или построены на машинном обучении.

Цель исследования — разработка прототипа с применением Tomita-парсера для анализа атрибутивной тональности текстовых сообщений. В соответствии с целью исследования, определяются следующие задачи работы:

A. Исследовать возможности Tomita-парсера для извлечения тональной информации из текстовых сообщений.
B. Разработать правила для Томита-парсера для извлечения "эмоционально–окрашенных"слов из текстовых сообщений.
C. Разработать модуль на C#, извлекающий тональные слова из текстовых сообщений.
D. Тестирование разработанного модуля.

ГЛАВА 1. АНАЛИТИЧЕСКИЙ ОБЗОР

В данной главе будет проведено исследование понятия «тональный анализ»1.1 и его методов 1.2, проведено исследование основных возможностей Tomaita-парсера 2.2, проведено исследование доступных тональных словари и сделан выбор в пользу одного из них 2.2, а также выбран способ создания прототипа для анализа текстовых сообщений 1.5.

1.1. Понятие тонального анализа

Тональный анализ (англ. Sentiment analysis) — это область компьютерной лингвистики, которая занимается изучением мнений и эмоций в текстовых документах [7]. Тональность текста в целом определяется лексической тональностью составляющих его единиц и правилами их сочетания. Присвоение тональности может происходить по разным правилам:

- классификация по бинарной шкале (оценка может быть либо позитивной, либо негативной)
- классификация по многополосной шкале (дискретные значения из ограниченного промежутка)
- системы шкалирования (непрерывная шкала оценок, отрезкам на которой соответствуют словесные описания тональности)

В данной работе будет использоваться система шкалирования, так как она позволяет оценить тональность не округляя значение, полученное с помощью вычислений. Также в работе будут использоваться 4 варианта словесной оценки тональности:

- позитивная
- негативная
- нейтральная
- противоречивая

1.2. Исследование существующих методов анализа тональности текстовых сообщений

Так как запрос на тональный анализ текста актуален на данный момент [8], то существуют различные методы для его проведения. Каждый из способов имеет свои преимущества и недостатки, которые будут подробно описаны далее.

A. *Методы, основанные на машинном обучении*

Данные методы используют с ранее размеченные эталонные блоки и относят их к негативу или позитиву на основании полученного результата сравнения [3]. При таком подходе требуется большой объем вводных данных. Преимуществами данных методов в гибкости, так как система без труда переобучается, и в высоких показателях полноты извлечения информации [5]. Недостатками же являются [4]:

- трудоемкость подготовки и обучения (требуется большой объем текстов)
- сложность выявления и исправления ошибки

B. *Методы, основанные на правилах и словарях*

Данные методы основываются на заранее составленных тональных словарях и правилах с использованием лингвистического анализа [2].Тональные словари должны содержать в себе достаточное количество слова для использования в исследуемой предметной области. Правила могут представлять из себя регулярные выражения или основываться на связи слов в тексте. Чтобы проанализировать текстовое сообщение методом, основанном на правилах и словаре, необходимо сначала составить тональный словарь, затем присвоить всем словам тональность и после сделать вывод об общей тональности текста. Общая тональность может определяться как среднее арифметическое всех тональных значений. Преимуществом данных методов является точность. Среди недостатоков можно отметить следующие:

- трудоемкость составления словаря, достаточно полным для исследования нужной предметной области
- неоднозначность присвоения тональности (слово «громкий» может иметь положительную окраску, когда речь идет о громком голосе певицы, но негативную, когда речь идет о шуме)

- сложности определения тональности при наличии орфографических ошибок в тексте

Для работы был выбран метод, основанный на тональном словаре, но алгоритм присвоения тональности при этом был изменен. Данному методу было отдано предпочтение по следующим причинам:

- результат не зависит от используемых текстов
- высокая точность
- не требуется большой объем данных для получения удовлетворительного результата

Как было упомянуто ранее, алгоритм присвоения тональности подвергся модификации. Стандартно словам в текстовом сообщении присваивается определенная тональность, полученная при использовании тонального словаря, а затем путем арифметических операций получается некоторое среднее значение,по которому будет сделан вывод о тональности всего текста. При таком алгоритме высока вероятность в противоречивом тексте получить вывод, свидетельствующий об отсутвии тональности («нейтральная» тональность). Например, «Сегодня прекрасная погода, но ужасное настроение». Согласно первоначальному алгоритму слову «прекрасная» будет присвоена позитивная тональность, а слову «ужасное» негативная. Будет ли значение тональности этих в цифровом эквиваленте ровно противоположно сказать трудно, но скорее всего их разность все же будет близка к нулю. Таким образом, велик риск, что данному предложению будет присвоена нейтральная тональность, хотя автор дал яркую эмоционально оценку словам «погода» и «настроение». Чтобы избежать подобного результата можно анализировать текст, присваивая общую тональность атрибутам. Тогда в результате будет получено следующее: «настроение» - негативная тональность, «погода» - позитивная тональность. А общая тональность текста будет скорее противоречивой. Данный подход позволяет более точно оценить тональность текста и не проигнорировать эмоциональное отношение автора к объектам в этом тексте.

1.3. Исследование Tomita-парсера

Tomita-парсер - это инструмент, разработанный компанией Яндекс, позволяющий извлекать структурированные данные из текста на естественном языке. Вычленение фактов происходит при помощи контекстно-свободных грамматик и словарей ключевых слов. Парсер позволяет писать свои грамматики и добавлять словари для нужного языка [10]. Парсер был назван в честь японского ученого Масару Томита, который разработал алгоритм для анализа текста на естественном языке. Именно этот алгоритм лежит в основе работы Томита-парсера. С помощью парсера можно выделять цепочки слов или факты по написанным пользователем шаблонам [11]. Парсер включает в себя три стандартных лингвистических процессора: токенизатор (разбиение на слова), сегментатор (разбиение на предложения) и морфологический анализатор (mystem). Файлы, используемые для работы с парсером приведены на рис.1.1.

Содержание	Формат	Примечания
config.proto — конфигурационный файл парсера. Сообщает парсеру, где искать все остальные файлы, как их интерпретировать и что делать.	Protobuf	Нужен всегда.
dic.gzt — корневой словарь. Содержит перечень всех используемых в проекте словарей и грамматик.	Protobuf / Gazetteer	Нужен всегда.
mygram.cxx — грамматика	Язык описания грамматик	Нужен, если в проекте используются грамматики. Таких файлов может быть несколько.
facttypes.proto — описание типов фактов	Protobuf	Нужен, если в проекте порождаются факты. Парсер запустится без него, но фактов не будет.
kwtypes.proto — описания типов ключевых слов	Protobuf	Нужен, если в проекте создаются новые типы ключевых слов.

Рис.1.1. Файлы для работы с Томита-парсером

Согласно информации на рис.1.1 минимальный набор файлов для работы с парсером - файл конфигурации и корневой словарь. После запуска парсера формируется как минимум один файл PrettyOutput.html, который в табличном виде содержит в себе цепочки, которые выделила грамматика. На рис.1.2 пример вывода полученных парсером цепочек. Грамматики парсера работают следующим образом: предложения представляют собой цепочки, из них выделяются подцепочки, а подцепочки интерпретируются в разбитые по полям факты.

Чтобы извлечь из текста подцепочки, представляющие собой существительные, нужно записать следующее правило грамматики:

$$S- > Noun; \tag{1.1}$$

С помощью этого правила из уже знакомого предложения «Сегодня прекрасная погода, но ужасное настроение» будут извлечены подцепочки в нормализованном виде: «сегодня», «погода» и «настроение».

Сегодня прекрасная погода , но ужасное настроение **EOS**

Text	Type
сегодня	TAuxDicArticle [грамматика]
погода	TAuxDicArticle [грамматика]
настроение	TAuxDicArticle [грамматика]

Рис.1.2. Цепочки, полученные в результате работы парсера

С помощью Томиты-парсера становится возможным извлечение из текстовых сообщений атрибутов и тональных слов, к ним относящихся.

1.4. Исследование тональных словарей

Так как для исследования был выбран метод, основанный на работе с тональными словарями, то необходимо было выбрать или составить самостоятельно словарь для дальнейшего использования. Необходимо определить требования к словарю, чтобы среди обилия вариантов выбрать наиболее подходящий вариант. Для определения тональности атрибутов было решено использовать прилагательные, относящиеся к этим атрибутам. Прилагательное - это часть речи, обозначающая качество, свойство или принадлежность предмета и изменяющаяся по падежам, числам и родам. Именно с помощью прилагательных в большинстве случаев дается эмоциональная оценка объектам.

По этой причине в тональном словаре должен быть широкий набор общеупотребительных прилагательных и существительных. Для исследования не была выбрана узкая область, предполагается, что работа будет происходить с текстами, содержащими в себе бытовую лексику. Для интерпретации оценки тональности ранее была выбрана методика, основанная на системе шкалирования, поэтому необходимо, чтобы словарь содержал в себе числовую оценку тональности из некоторого непрерывного диапазона.

После продолжительного поиска было найдено 3 возможных варианта тональных словарей для дальнейшего использования:

- тональный словарь PolSentiLex. Достоинства: имеет большую базу слов, проект реализован специалистами Лаборатории интернет-исследований НИУ ВШЭ – СПб. Недостатки: имеет дискретную систему оценки тональности (-2, -1, 0, 1, 2).
- словарь оценочных слов и выражений русского языка РуСентиЛекс. Достоинства:содержит в себе фразы и выражения, а также сленг, всего в базе более 12 тысяч слов и выражений. Недостатки: тональность определяется словесно (позитивная, негативная, нейтральная или неопределенная оценка)
- Тональный словарь русского языка КартаСловСент. Достоинства: включает в себя слова и выражения русского языка, снабжённые тональной меткой («положительное», «отрицательное», «нейтральное») и скалярным значением силы эмоционально-оценочного заряда из непрерывного диапазона [-1, 1], содержит в себе более 46 тысяч записей. Недостатки: для данной задачи не обнаружено.

Наиболее подходящим для данных задач словарем оказался словарь КартаСловСент. Фрагмент файла словаря можно увидеть на рис.1.3.

```
term;tag;value;pstv;ngtv;neut;dunno;pstvNgtvDisagreementRatio
абажур;NEUT;0.08;0.185;0.037;0.58;0.198;0.0
аббатство;NEUT;0.1;0.192;0.038;0.578;0.192;0.0
аббревиатура;NEUT;0.08;0.196;0.0;0.63;0.174;0.0
абзац;NEUT;0.0;0.137;0.0;0.706;0.157;0.0
абиссинец;NEUT;0.28;0.151;0.113;0.245;0.491;0.19
абитуриент;NEUT;0.23;0.235;0.049;0.5;0.216;0.0
абитуриентка;NEUT;0.34;0.294;0.029;0.461;0.216;0.0
абонемент;NEUT;0.18;0.232;0.056;0.56;0.152;0.0
абонементный;NEUT;0.19;0.21;0.07;0.5;0.22;0.0
абонент;NEUT;0.0;0.075;0.075;0.675;0.175;0.0
абонентный;NEUT;0.0;0.073;0.0;0.61;0.317;0.0
абонентский;NEUT;0.0;0.071;0.0;0.822;0.107;0.0
абордаж;NGTV;-0.45;0.07;0.33;0.38;0.22;0.0
абордажный;NEUT;-0.19;0.137;0.171;0.384;0.308;0.11
абориген;NEUT;-0.02;0.063;0.167;0.624;0.146;0.0
```

Рис.1.3. Тональный словарь русского языка КартаСловСент

В первой строчке рис.1.3 содержатся названия полей:

- term - слово или словосочетание
- tag - метка тональности: PSTV («положительное»), NGTV («отрицательное»), NEUT («нейтральное»)
- value - скалярное значение эмоционально-оценочного заряда из непрерывного диапазона [-1, 1], где +1 соответствует входам с максимально положительной окраской, -1 — входам с максимально отрицательной окраской, 0 — входам с нейтральной окраской (то же, что отсутствие окраски)
- а также другие поля, характеризующие доли голосов за определенную тональности

В данном словаре уже используется система для перевода числового значения тональности в словесный: $[-1, -0.3]$ - негативная тональность, $[0.5, 1]$ - позитивная тональность, остальное - нейтральная. Эта система будет использоваться в дальнейшем для всего приложения.

1.5. Выбор способа реализации

Для запуска Томиты-парсера используется командная строка, а для просмотра полученных результатов html файл, хоть в командной строке после завершения работы и появляется строка xml формата с этими же результатами. Это сделано потому, что воспринимать информацию человеку удобнее, когда она визуализирована, например, в виде таблицы. По этой причине разрабатываемый прототип должен иметь графический интерфейс и простой дизайн. Для создания приложений с графическим интерфейсом отлично подходит технология Windows Forms. Разработка приложения будет производиться с использованием языка C#.

ГЛАВА 2. РАЗРАБОТКА ПРОТОТИПА

В данной главе последовательно описывается разработка прототипа для тонального анализа текста. В первом параграфе главы 2.1 описан краткий алгоритм работы прототипа. В параграфе 2.2 подробно описан процесс разработки правил грамматики для Томита-парсера, а также содержание других файлов для работы с парсером. В параграфе 2.3 описан процесс присвоения тональности извлеченным парсером словам и формирование окончательного результата по анализу атрибутивной тональности текста. Параграф 2.4 содержит в себе описание внешнего вида приложения, а точнее - описание графического интерфейса. Тестирование разработанного приложения производится в параграфе 2.5 на основе сравнения ожидаемых результатов и полученных.

2.1. Краткое описание алгоритма

После запуска приложения пользователь вводит текст. Далее этот текст обрабатывается приложением в несколько этапов, которые представлены в коде приложения методами:

- метод GetText() удаляет старый файл с текстом (если есть), записывает данные введенные от пользователя в новый файл с названием test.txt
- метод makeCodeTomita(int mode) в зависимости от аргумента формирует first.cxx — файл грамматики двумя разными способами
- метод StartTomita() осуществляет запуск парсера
- метод ReadFacts() отвечает за получение фактов из парсера и их обработку
- метод UpdateGrid() формирует вывод результатов обработки фактов пользователю

2.2. Разработка правил для Tomita-парсера

В первой главе были изучены основные принципы работы с Томита-парсером. Как уже упоминалось в параграфе ,для определения тональности необходимо получить атрибуты и прилагательные к ним относящиеся. Атрибутами будут выступать существительные. Таким образом, правила для парсера должны извлекать цепочки, состоящие из существительного и прилагательного (прилагательных).

Файл грамматики был создан с именем first.cxx. Для того, чтобы парсер знал об этом файле, в файле словаря необходимо написать следующую команду (см. рис.2.1).

```
TAuxDicArticle "грамматика"
{
    key = { "tomita:first.cxx" type=CUSTOM }
}
```

Рис.2.1. mydic.gzt

В конфигурационном файле config.proto нужно добавить грамматику (см. рис.2.2).

```
Articles = [
{ Name = "грамматика" } ]
```

Рис.2.2. config.proto

Далее необходимо написать правила для Томиты-парсера. У атрибута может быть одно существительное, а может быть несколько разделенных запятыми и/или союзами. Извлечение фактов будет проходить в два этапа: сначала нужно извлекать цепочку, содержащую в себе атрибут и прилагательные, а затем из группы прилагательных выделять одиночные слова.

Правила можно прописать в одном файле, но тогда есть риск, что некоторые группы прилагательных будут разбиваться неверно. Например, если на вход подается текст «мой рабочий костюм» , то сначала выделится атрибут «костюм» и группа слова «мой рабочий» . Затем, парсер будет запущен снова и несмотря на наличие правил для извлечения только прилагательных, парсер извлечет атрибут «рабочий» , так как в данном словосочетании, действительно, складывается впечатление, что «рабочий» - это существительное. Для того, чтобы этого избежать, было принято решение формировать файл грамматики first.cxx во время работы программы. В таком случае сначала будут извлечены все цепочки содержащие атрибуты, затем правила в грамматике поменяются и на вход парсеру будут подаваться группы прилагательных, которые будут разбиваться по одному по новым правилам. Для извлечения цепочек существительное + прилагательные были написаны следующие команды (см. рис.2.3)

В строке 1 (рис.2.3) формируется запись, представляющая собой одно прилагательное (Adj - прилагательное, причастие, порядковое числительное или

```
1 AdjCoord->Adj;
2 AdjCoord->AdjCoord <gnc-agr[1], rt> ',' AdjCoord <gnc-agr[1]>;
3 AdjCoord->AdjCoord <gnc-agr[1], rt> 'и' AdjCoord <gnc-agr[1]>;
4 S->Noun<gnc-agr[1]> interp(Fact.Noun) AdjCoord<gnc-agr[1]>+
    interp(Fact.Adj::norm = "m,sg");
5 S->AdjCoord <gnc-agr[1]> +interp(Fact.Adj::norm = "m,sg") Noun
     <gnc-agr[1]> interp(Fact.Noun);
```

Рис.2.3. first.cxx

```
1 S->Adj interp(FactAdj.A::norm = "m,sg");
```

Рис.2.4. first.cxx

местоименное прилагательное). В строке 2 (рис.2.3) обрабатывается случай, когда прилагательные разделены запятыми. <gnc-agr[1]> указывает на согласование по роду, числу и падежу. В строке 3 (рис.2.3) аналогичным образом обрабатывается случай, когда прилагательные разделены союзом «и». В четвертой строке S является вершиной формируемой цепочки. Цепочка будет формироваться так: существительное Noun, согласованное по роду, числу и падежу с одной или более AdjCoord записью. На количество «один или более» указывает «+» после AdjCoord. Фрагмент «interp(Fact.Adj::norm = «m,sg»)» указывает на извлечение слова, которое будет нормализовано: мужской род («m»), единственное число («sg»). Нормализация нужна для поиска по словарю, в котором слова представлены в единственном числе, в мужском роде. Команда «interp» позволяет интерпретировать факты, то есть распределить подцепочки из выделенной грамматикой цепочки по полям факта «Adj». В строке 5 (рис.2.3) обрабатывается случай, если прилагательные стоят перед существительным.

Для извлечения из текста только прилагательных в файле first.cxx нужно сформировать следующую команду:

Команда (рис.2.4) извлекает прилагательные и нормализует их по уже описанным правилам.

Также необходимо создать файл facttypes.proto и записать туда информацию об извлекаемых фактах:

В строках 1 и 6 (рис.2.5) задаются имена фактам. В строках 3, 4 и 8 указывается, что поля фактов это строки с именами Noun, Adj, A (эти имена указываются в интерпретации в файле грамматики first.cxx (рис.2.3 , рис.2.4) {Имя факта}.{Имя поля}).

```
message Fact: NFactType.TFact
{
    required string Noun = 1;
    required string Adj = 2;
}
message FactAdj: NFactType.TFact
{
    required string A = 1;
}
```

Рис.2.5. facttypes.proto

```
Facts = [
    { Name = "Fact" },
    { Name = "FactAdj" }
]
```

Рис.2.6. config.proto

Для того, чтобы было удобнее программно получать информацию от парсера, можно формировать файл output.txt. Чтобы сформировать данный файл, необходимо в config.proto добавить следующий фрагмент кода (рис.2.6):

Данный код (рис.2.6) указывает на сформированные ранее правила интерпретации цепочек.

Текст, получаемый парсером на обработку, хранится в файле test.txt и представляет собой текстовое сообщение, записанное на естественном языке. Пример извлечения фактов парсером из текста «Сегодня прекрасная погода, но ужасное и мерзкое настроение» см. на рис.2.7 с применением грамматики (рис.2.3).

```
Сегодня прекрасная погода , но ужасное и мерзкое настроение
        Fact
        {
                Noun = погода
                Adj = прекрасный
        }
        Fact
        {
                Noun = настроение
                Adj = ужасный и мерзкий
        }
```

Рис.2.7. Файл output.txt

Были извлечены атрибуты «погода» и «настроение», так как они имели согласованные с ними прилагательные. Теперь согласно алгоритму на вход парсеру

пойдут поочередно строки «прекрасный» и «ужасный и мерзкий». Первую строку проверять нет необходимости (понятно, что это одно слово и оно уже может использоваться для поиска в тональном словаре), чтобы сообщить это программе будет введена проверка на наличие пробела в строке (как минимум один пробел между прилагательными будет, если прилагательных 2 или более). Результат анализа парсером строки «ужасный и мерзкий» см. на рис.2.8.

```
ужасный и мерзкий
        FactAdj
        {
                A = ужасный
        }
        FactAdj
        {
                A = мерзкий
        }
```

Рис.2.8. Файл output.txt

Прилагательные были получены верно в нормализованном виде, значит теперь можно переходить к этапу назначения тональности атрибутам. Для того, чтобы запускать парсер из приложения был разработан метод StartTomita(), который представлен на листинге (рис.2.9). Данный метод позволяет открыть командную строку без создания окна, перейти в нужную директорию и запустить парсер командой "tomitaparser.exe config.proto"(строка 11 (рис.2.9)).

Для чтения данных после первого запуска парсера (когда извлекаются атрибуты + прилагательные) был разработан метод ReadFacts(). Данный метод вычленяет слова из файла, а затем помещает их в список списков *List<List<string>> words* таким образом, чтобы данные были похожи на таблицу (см. рис.2.10):

На рис.2.10 внесены данные, которые бы были получены из текста «Сегодня прекрасная погода, но ужасное и мерзкое настроение» на основе двух этапов работы с парсером. В первый этап были бы получены известные факты (рис.2.7), во второй этап строки с прилагательными проверяются на количество слов, если нужно, разбиваются на отдельные слова. Номер цепочки определяется по существительному (атрибуту). То есть все прилагательные, относящиеся к одному существительному будут иметь один номер. Если атрибут встречается несколько раз, то такой подход позволит записать все прилагательные к одной цепочке, а значит, к одному атрибуту. Это позволит избежать ситуации, при которой атрибут, являющийся одним и тем же существительным, будет рассматриваться как несколько разных атрибутов.

Листинг2.1

StartTomita()

```
1         private void StartTomita()
2         {
3             Process cmd = new Process();
4             cmd.StartInfo.FileName = "cmd.exe";
5             cmd.StartInfo.RedirectStandardInput = true;
6             cmd.StartInfo.RedirectStandardOutput = true;
7             cmd.StartInfo.CreateNoWindow = true;
8             cmd.StartInfo.UseShellExecute = false;
9             cmd.Start();
10            cmd.StandardInput.WriteLine(@"cd C:/tomita");
11            cmd.StandardInput.WriteLine(@"tomitaparser.exe
                 config.proto");
12            cmd.StandardInput.Flush();
13            cmd.StandardInput.Close();
14            cmd.Close();
15        }
```

Рис.2.9. StartTomita()

Aa Слово	Adj/Noun	№ подцепочки	Тональность
Погода	Noun	1	
Прекрасный	Adj	1	
Настроение	Noun	2	
Ужасный	Adj	2	
Мерзкий	Adj	2	

Рис.2.10. Схематичное изображение структуры для хранения слов List<List<string> > words

2.3. Присвоение тональности атрибутам

Для удобного использования тонального словаря был разработан метод *getTonalDictionary();*, который запускается вместе с запуском приложения и переносит пары «слово» - «численное значение тональности» в *Dictionary<string, float> wordsTones*. После того, как введенный пользователем текст будет обработан парсером, каждому слову из *List<List<string> > words* будет присвоена тональность из *Dictionary<string, float> wordsTones*. После этого *List<List<string> > words* можно визуализировать как на рис.2.11:

Aa Слово	Adj/Noun	№ подцепочки	Тональность
Погода	Noun	1	0.03
Прекрасный	Adj	1	1.0
Настроение	Noun	2	0.02
Ужасный	Adj	2	-1.0
Мерзкий	Adj	2	-1.0

Рис.2.11. Схематичное изображение структуры для хранения слов List<List<string> > words

Если слово не было найдено в словаре тональности, то тогда ему присваевается значение 0.

После того как тональность из словаря была присвоена всем извлеченным парсером словам, нужно разобраться какая же тональность у атрибутов. Благодаря хранению номеров цепочек, посчитать тональность атрибута не составляет труда. Тональность каждого существительного *List<List<string> > words* суммируется с тональностью прилагательных, относящихся к той же цепочке, что и существительное (атрибут), а затем эта сумма делится на количество слов. То есть тональность атрибута вычисляется как среднее арифметическое значений его тональности и тональных слов (прилагательных). Также совершается попытка определить общею тональность текста. Если тональность отрицательная отличается от положительной меньше, чем на 10% и они не равны 0, то тогда тексту присваивается «противоречивая» тональность. Если положительная или отрицательная тональность равна нейтральной, тогда делается вывод о том, что тональность не нейтральная, а положительная/отрицательная. В противных случаях, тональность определяется по большинству атрибутов, имеющих ту или иную тональность.

2.4. Разработка графического интерфейса

Согласно сделанным ранее утверждениям в Главе 1 1.5, прототип, проводящий тональный анализ текста, должен иметь графический интерфейс. Для создания интерфейса использовались следующие компоненты windows forms: Button, Label, GroupBox, RichTextBox и DataGridView. На рис.2.12 проилюстрировано приложение с указанием на используемые компонены.

На рис.2.13 демонстрация работы созданного прототипа для анализа тональной информации с использованием Tomita-парсера и тонального словаря.

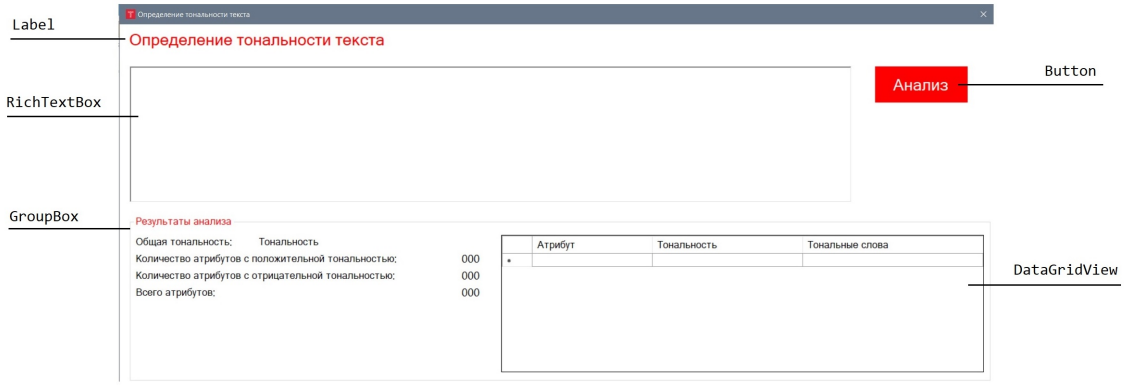

Рис.2.12. Скриншот интрефейса разработанного приложения с подписями компонентов

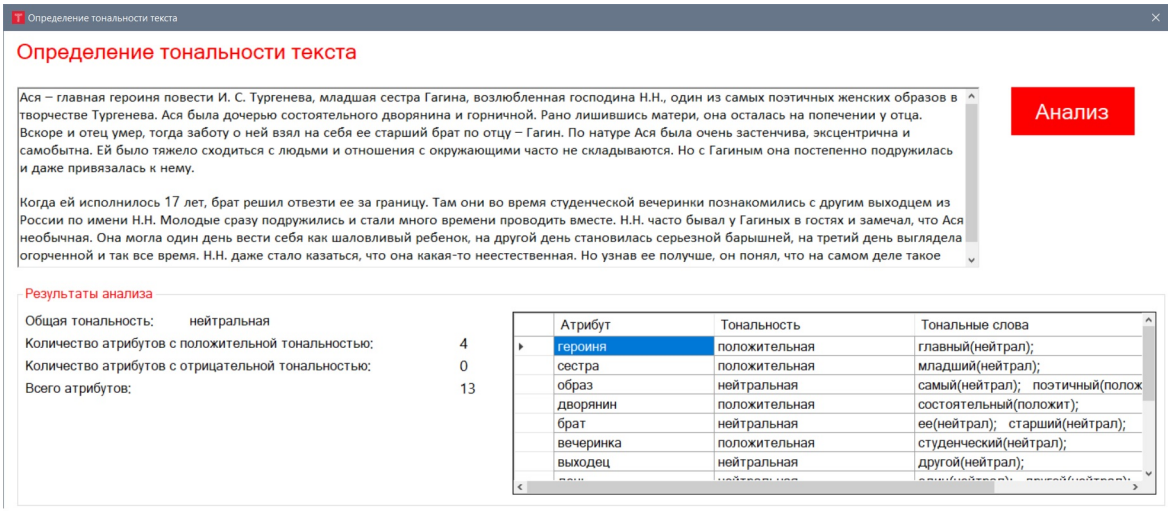

Рис.2.13. Демонстрация работы приложения

2.5. Тестирование разработанного прототипа

Тестирование разработанного прототипа будет присходить путем сравнения ожидаемых результатов и полученных. Для тестирования работы приложения выделены следующие ситуации:

- однородные прилагательные перед существительным
- неоднородные прилагательные перед существительным
- одно и то же существительное употребляется в тексте множество раз с разными прилагательными
- наличие прилагательных, подвергшихся процессу субстантивации
- отсутствие атрибутов, имеющих тональные слова
- наличие в тексте атрибутов имеющих как позитивную тональность, так и негативную (примерно в равном количестве)

Сначала было осуществлено тестирование при наличии однородных прилагательных перед существительным. Члены предложения, которые отвечают на

один и тот же вопрос и относятся к одному и тому же определяемому слову, называются однородными. [6]. Однородные члены предложения отделяются друг от друга запятой или союзом.Рассмотрим предложение: «По натуре Ася застенчивая, эксцентричная и самобытная девушка». Очевидно, что в данном предложении прилагательные «застенчивая, эксцентричная и самобытная» относятся к атрибуту «девушка». На рис.2.14 продемонстрирован результат работы приложения с этим предложением.

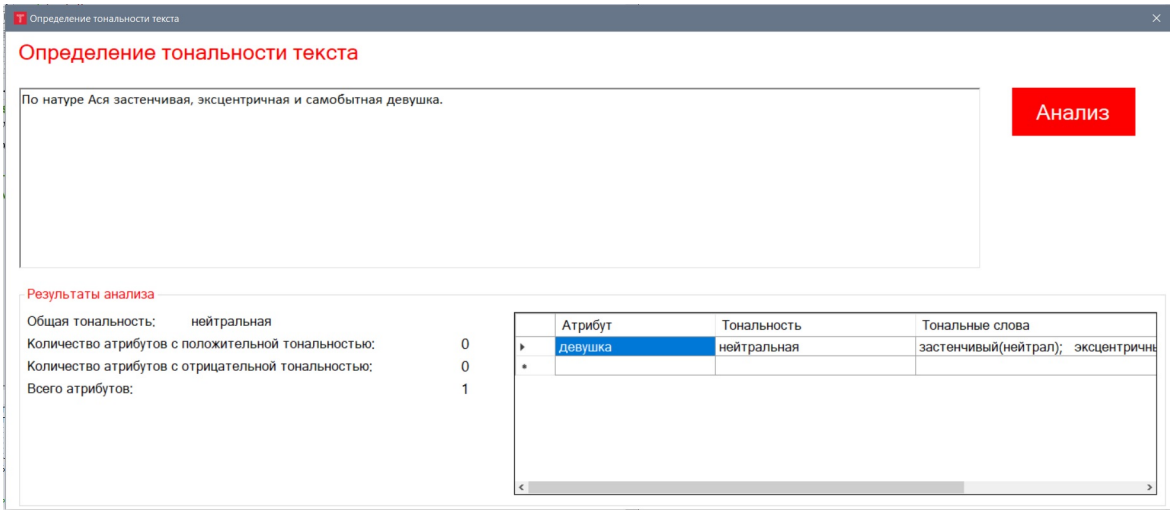

Рис.2.14. Результат работы приложения при начилии однородных прилагательных перед существительным

На рис.2.15 приведен полный список извлеченных прилагательных. Все слова, которые должны быть извлечены для атрибута «девушка» были извлечены.

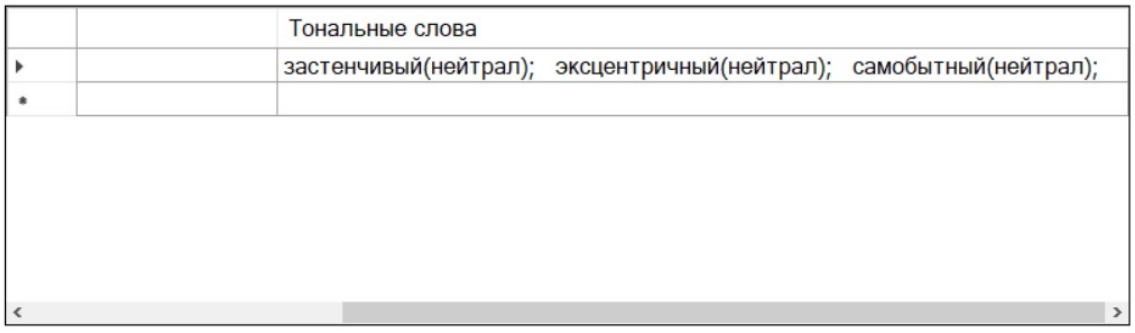

Рис.2.15. Полный список тональных слов, полученных в результате работы приложения при начилии однородных прилагательных перед существительным

Этот этап тестирования можно считать пройденным успешно.

Следущющим этапом будет произведено тестирование при наличии неоднородных прилагательных перед существительным. Неоднородные члены предложения не произносятся с перечислительной интонацией, между ними обычно нельзя поставить союз и, в отличии от однородных членов, рассмотренных в **??** [1].

Пример предложения, содержащего неоднородные прилагательные: «Ася – главная героиня повести И. С. Тургенева, один из самых поэтичных женских образов в творчестве Тургенева.». В таком предложении можно выделить атрибуты «героиня» и «образ». Неоднородные прилагательные относятся к атрибуту «образ»: «самый», «поэтичный», «женский». Прилагательные идут подряд и разделены пробелами. На рис.2.16 продемонстрирован результат работы приложения с этим предложением.

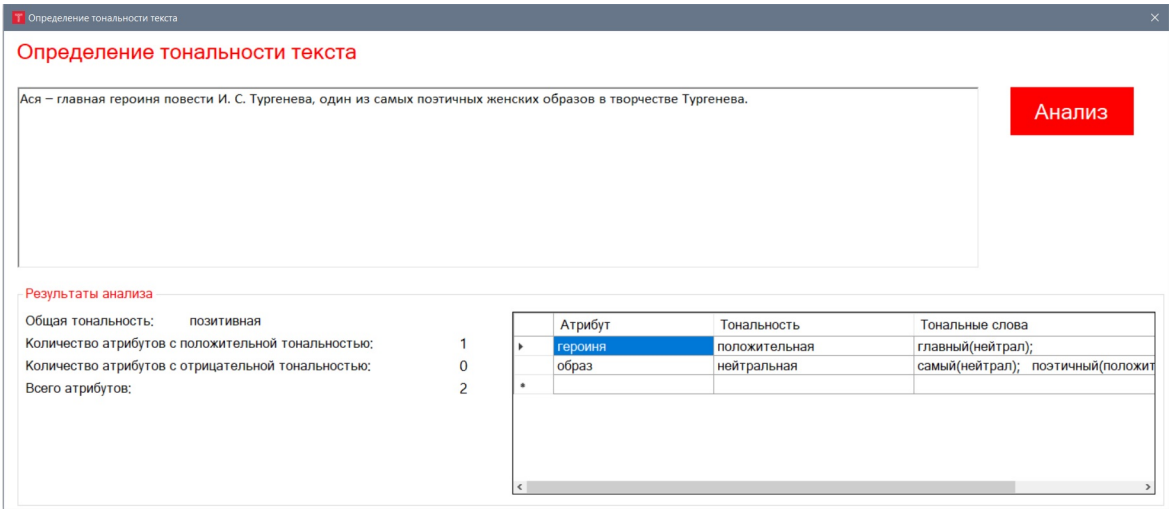

Рис.2.16. Результат работы приложения при начилии неоднородных прилагательных перед существительным

На рис.2.17 приведен полный список извлеченных прилагательных. Все слова, которые должны быть извлечены для атрибута «образ» были извлечены, на рис.2.17 они выделены синим цветом.

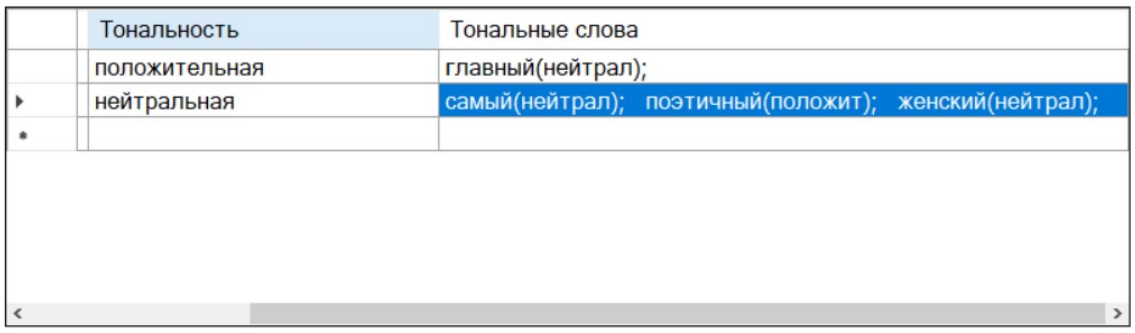

Рис.2.17. Полный список тональных слов, полученных в результате работы приложения при начилии неоднородных прилагательных перед существительным

Далее будет проведено тестирование при многократном упоминании атрибута в тексте. В данном блоке тестирования необходимо проверить действительно ли все прилагательные, относящиеся к одному и тому же существительному извлекаются? Меняется ли тональность атрибута, если какое-то тональное слово повторяется? Текст для тестирования: «Когда книга интересная, то читать ее легко

и приятно. Этого нельзя сказать о скучных книгах. Во время чтения интересной книги можно потерять счет времени.». Атрибутом, имеющим тональные слова, в тексте выступает слово «книга». Это слово употребляется как в единственном, так и множественном числе. Также к атрибуту относятся прилагательные «скучный» (один раз) и «интересный» (два раза). Очевидно, что слово «скучный» имеет негативную тональность, в то время как «интересный», наоборот, позитивную. Получается, у атрибута «книга» должна быть позитивная тональность, так как позитивных тональных слов в два раза больше, чем негативных. На рис.2.18 демонстрация результата работы приложения с упомянутым текстом.

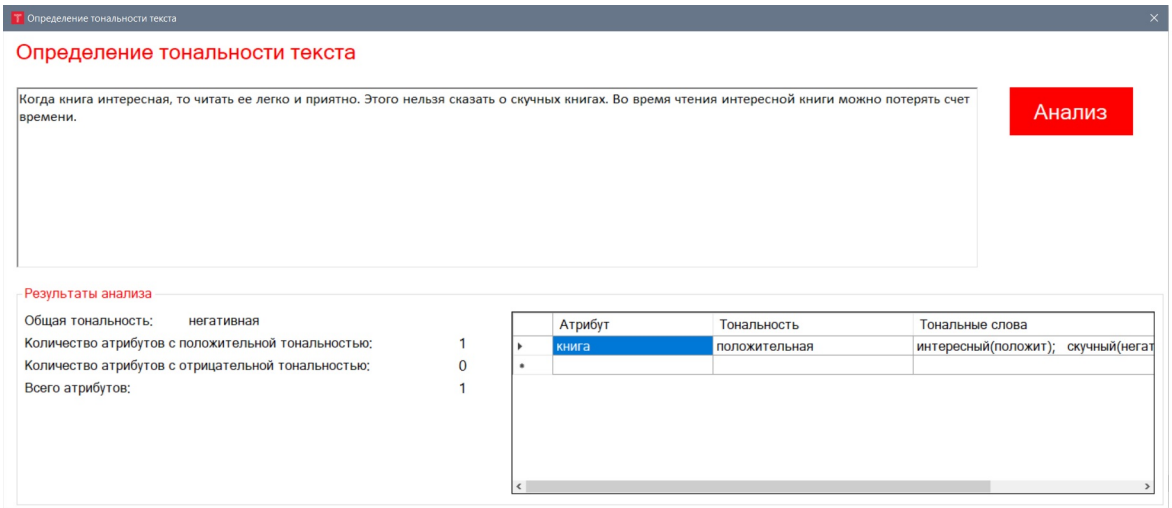

Рис.2.18. Тестирование при многократном упоминании атрибута в тексте

На рис.2.19 приведен полный список извлеченных тональных слов. Прилагательное «интересный» приведен в списке дважды и согласно рис.2.18 тональность атрибута была определена как положительная.

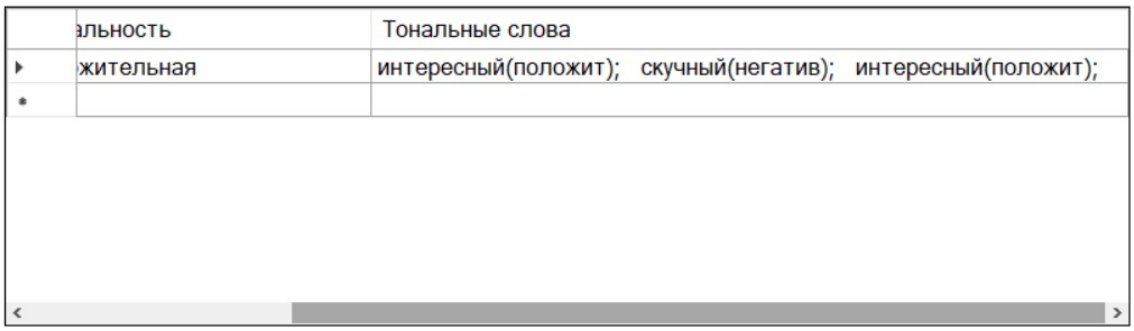

Рис.2.19. Полный список тональных слов, полученных в результате работы приложения при многократном упоминании атрибута в тексте

Следовательно, все тональные слова атрибуты были получены и учтены при расчете общей тональности атрибута.

Следующим этапом проводится тестирование при наличии прилагательных, подвергшихся процессу субстантивации. *Субстантивация* (от лат. substantivum—существительное) - это переход слов других частей речи в разряд имен существительных [9]. Например, «больной поправился» - субстантивация прилагательного «больной». Важно, чтобы при работе приложения прилагательные, которые подвержены субстантивации, не определялись как существительные, если они относятся к атрибутам. Так как процесс получения прилагательных происходит в два этапа: сначала получение парсером из начального текста цепочек существительное + прилагательные, к нему относящиеся, а потом на вход парсеру подается строка с извлеченными прилагательными, то важно, чтобы прилагательные во время второго запуска парсера оставались прилагательными для парсера. Теститирование будет проводиться на следующем тексте: «Уставший рабочий сел отдыхать на лавочку. Он отряхнул свой рабочий комбинезон и посмотрел вдаль.». В этом тексте слово «рабочий» выступает в роли как существительного, так и прилагательного. Но очевидно, что атрибутами, имеющими тональные слова, являются слова «рабочий» и «комбинезон». Демонстрация результата запуска приложения с вышеупомянутым текстом представлена на рис.2.20.

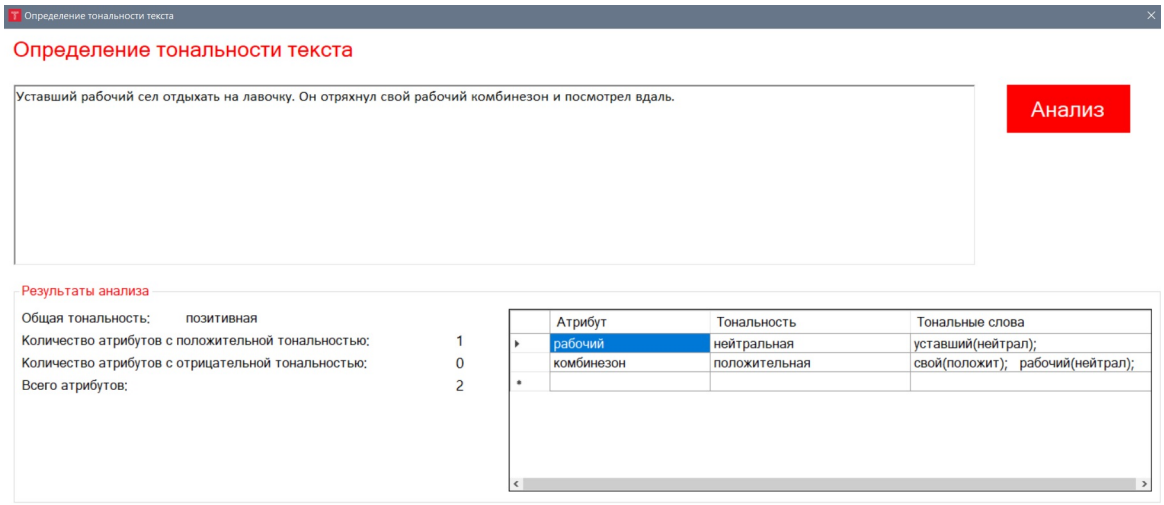

Рис.2.20. Тестирование при наличии прилагательных, подвергшихся процессу субстантивации

Приложение корректно обработало текст и правильно к атрибуту существительное,а прилагательное, как и ожидалось, было записано в тональные слова.

Далее проведено тестирование при отсутствии атрибутов, имеющих тональные слова. При отсутствии в тексте атрибутов, имеющих тональные слова, необходимо сообщить пользователю об этом. Демонстрация работы программы при вводе текста «Сегодня цветок завял», не содержащего в себе прилагательных, представлена на рис.2.21.

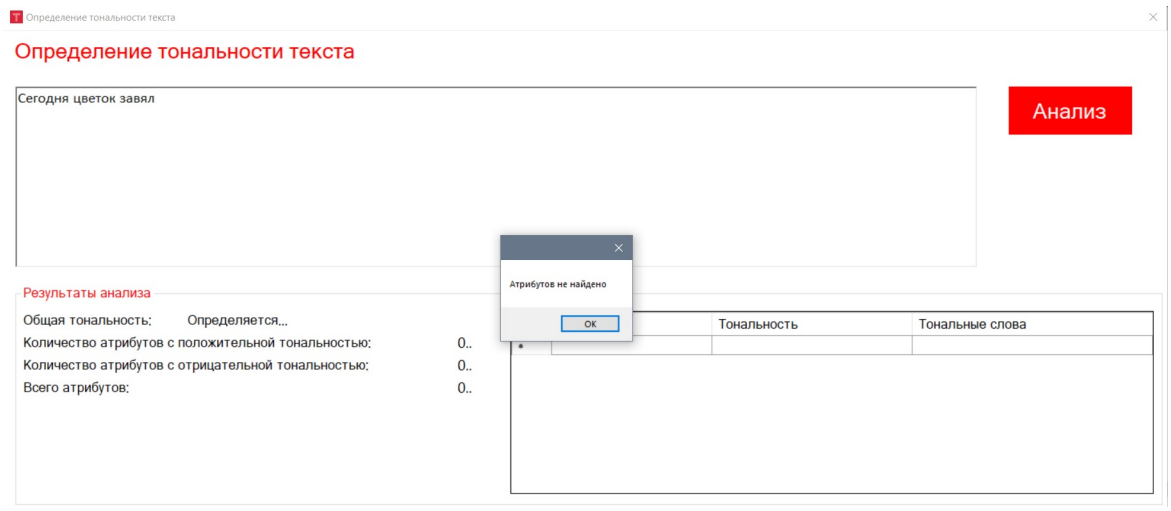

Рис.2.21. Тестирование при отсутствии атрибутов в тексте

Как видно на рис.2.21, после нажатия на кнопку «Анализ» появляется Message Box, уведомляющий о том, что атрибутов не найдено. После нажатия на «ок» тексту присваивается нейтральная тональность - см. рис.2.22

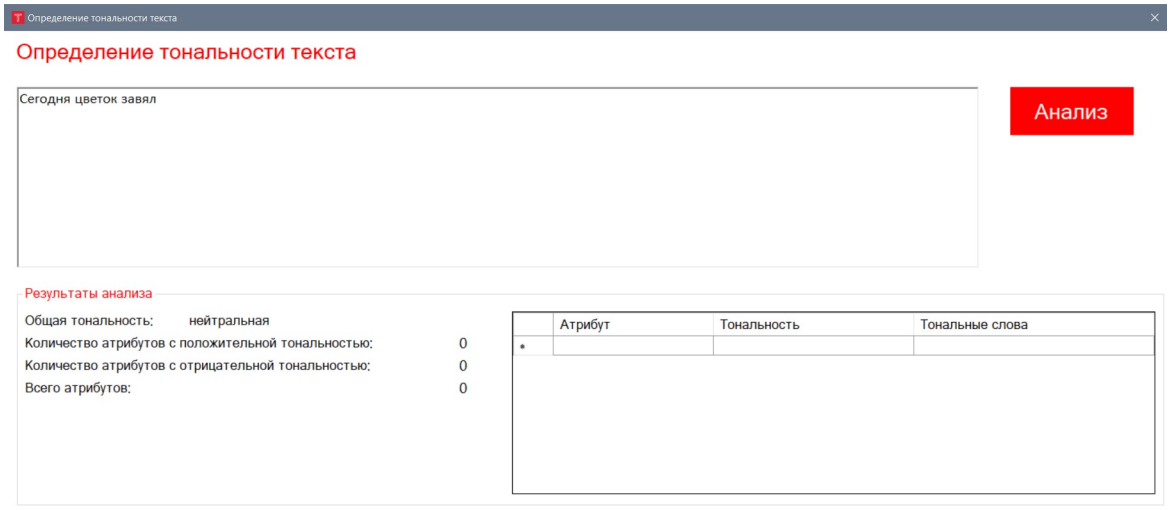

Рис.2.22. Определение тональности при отсутствии атрибутов в тексте

Следовательно, при отсутствии атрибутов, имеющих тональные слова, приложение уведомляет пользователя и присваивает тексту нейтральную тональность, - это являлось желательным результатом.

Далее проводится тестирование при наличии атрибутов, имеющих противоположную тональность. Тестирование будет проводиться при использования текста, упоминавшйся ранее при исследовании Томита-парсера 2.2: «Сегодня прекрасная погода, но ужасное настроение». Тональность данного предложения затруднительно определить однозначно. Поэтому желаемым результатом является корректная оценка тональности каждого из атрибутов («погода» - позитивная

тональность, «настроение» - негативная) и вывод о противоречивой тональности всего текста. Результат тестирования можно наблюдать на рис.2.23.

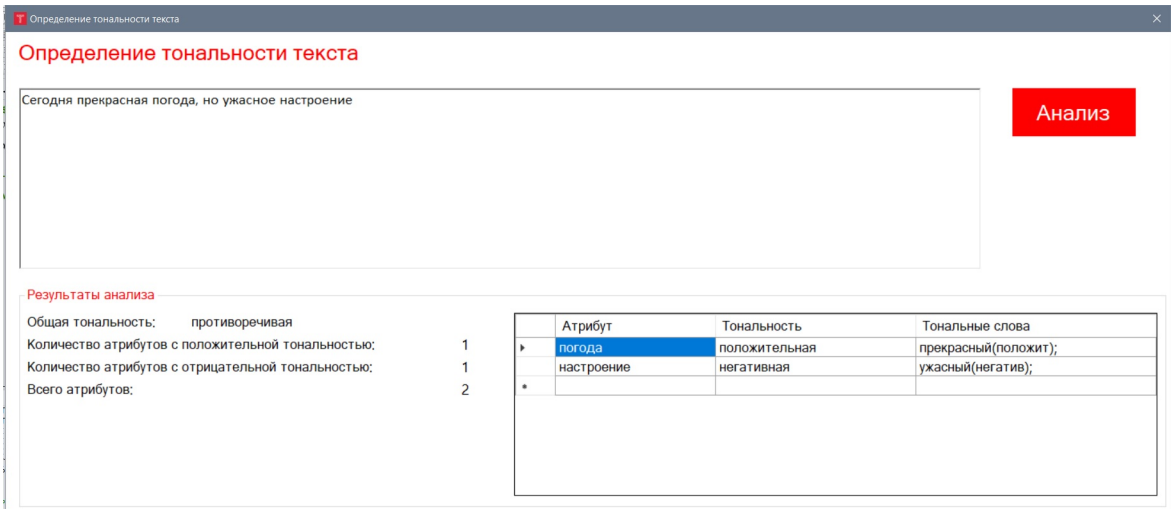

Рис.2.23. Тестирование при отсутствии атрибутов в тексте

Как видно на рис.2.23 полученные результаты полностью совпали с желаемыми.

Согласно тестированию, которое заключалось в сопоставлении желаемых результатов и получаемых, приложение успешно справляется с задачей определения тональности атрибутов и имеет гибкую систему оценивания, что позволяет максимально точно оценить тональность текста. Следовательно, результаты тестирования удовлетворительные по каждому из рассматриваемых пунктов.

ЗАКЛЮЧЕНИЕ

Анализ тональности имеет важное практическое применение и поиск решений для его реализации имеет большой интерес от представителей самых разных отраслей. Существующие на данный момент методы оценки тональности текста имеют ряд недостатков, поэтому попытка разработать свой метод была обоснована. Использованный в работе Томита-парсер имеет большую функциональность. Благодаря существующей документации [11], которая включает в себя как текстовые объяснения, так и видеоролики, разработка правил для работы с парсером не представляет существенных сложностей. Для достижения поставленной цели, а именно - разработка прототипа с применением Tomita-парсера для анализа атрибутивной тональности текстовых сообщений были пройдены следующие этапы:

- рассмотрены методы оценки тональности, выявлены их достоинства и недостатки
- разработан метод для оценки тональности на основе рассмотренных методов
- разработан прототип в виде оконного приложения

Цель была достигнута, что подтверждается успешно проведенным тестированием разработанного прототипа. Для улучшения разработанного приложения можно разработать правила для Томиты-парсера, позволяющего извлекать слова других частей речи (например, наречия), чтобы повысить полноту оценивания тональности текста.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] *2.3.2. Однородные и неоднородные определения*. URL: https://licey.net/free/4-russkii_yazyk/40-kurs_russkogo_yazyka_sintaksis_i_punktuaciya/stages/713-232_odnorodnye_i_neodnorodnye_opredeleniya.html (дата обращения: 24.06.2021).

[2] *Анализ тональности текста*. URL: https://clck.ru/V9Dn9 (дата обращения: 06.06.2021).

[3] *Анализ тональности текста с использованием методов машинного обучения*. URL: http://ceur-ws.org/Vol-2233/Paper_8.pdf (дата обращения: 06.06.2021).

[4] *Извлечение объектов и фактов из текстов в Яндексе. Лекция для Малого ШАДа*. URL: https://habr.com/ru/company/yandex/blog/205198/ (дата обращения: 06.06.2021).

[5] *Классификация текстов и анализ тональности*. URL: https://clck.ru/SNgQv (дата обращения: 06.06.2021).

[6] *Когда между прилагательными ставится запятая?* URL: https://pishugramotno.ru/punktuacia/kogda-mezhdu-prilagatelnymi-stavitsya-zapyataya (дата обращения: 24.06.2021).

[7] *Обучаем компьютер чувствам*. URL: https://habr.com/ru/post/149605/ (дата обращения: 26.06.2021).

[8] *Системная социология: Opinion Mining*. URL: https://www.isras.ru/index.php?page_id=1024 (дата обращения: 06.06.2021).

[9] *Словарь лингвистических терминов. Субстантивация*. URL: https://classes.ru/grammar/114.Rosental/17-s-3/html/unnamed_68.html (дата обращения: 24.06.2021).

[10] *Томита-парсер*. URL: https://yandex.ru/dev/tomita/ (дата обращения: 11.06.2021).

[11] *Томита-парсер учебник*. URL: https://yandex.ru/dev/tomita/doc/tutorial/concept/about.html (дата обращения: 06.06.2021).

Приложение 1

Программный код скриптов на C#

Листинг 1.1

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Tomita
{
    public partial class Form1 : Form
    {
        private RichTextBox textBox;
        private DataGridView dataGrid;
        private Label numberPos;
        private Label numberNeg;
        private Label numberAll;
        private Label answer;

        Dictionary<string, float> tones = new Dictionary<
            string, float>();
        List<List<string>> words = new List<List<string>>();
        int numNoun = 0;
        int numRows = 0;
        int numEx;
        bool isFirstTime;

        Dictionary<string, float> wordsTones = new Dictionary<
            string, float>();
        public Form1()
        {
```

```
36              InitializeComponent();
37              textBox = this.richTextBox;
38              dataGrid= this.dataGridView1;
39              getTonalDictionary();
40              numberPos = this.labelPos;
41              numberNeg = this.labelNeg;
42              numberAll = this.labelAll;
43              answer = this.labelAns;
44          }
45          private void getAnswer(int all, int pos, int neg, int
                neut)
46          {
47              float d = (float) neut / (float) all;
48              float d1 = Math.Abs((float)pos - (float)neg) / all
                    ;
49              if (pos == neut && pos!=0) answer.Text = "позитивн
                    ая";
50              else if  (neg== neut) answer.Text = "негативная";
51              else
52              if ( d>=0.5) answer.Text = "нейтральная";
53              else
54                  if ((d1 <=0.1)) answer.Text = "противоречивая
                        ";
55              else if (pos>neg) answer.Text = "позитивная";
56              else answer.Text = "негативная";
57              if (pos == 0 && neg == 0 && neut == 0) answer.Text
                    = "нейтральная";
58          }
59          private void button_Click(object sender, EventArgs e)
60          {
61
62
63              numberPos.Text = "0..";
64              numberNeg.Text = "0..";
65              numberAll.Text = "0..";
66              answer.Text = "Определяется...";
67              words.Clear();
68              wordsTones.Clear();
69              dataGrid.Rows.Clear();
70              numNoun = 0;
71              numRows = 0;
72              isFirstTime = true;
73
74
75              GetText();
```

```
76              makeCodeTomita(0);
77              StartTomita();
78              ReadFacts();
79              UpdateGrid();
80
81              Console.WriteLine("words.Count    " + words.Count)
                    ;
82              Console.WriteLine("wordsTones.Count    " +
                    wordsTones.Count);
83          }
84
85
86          private void UpdateGrid()
87          {
88
89              int i = 0; float sum;int j = 0; int counter = 0;
90              StringBuilder adjStr= new StringBuilder();
91
92
93              int numNEG = 0;
94              int numPOS = 0;
95              int numNEUT = 0;
96
97              while (i<words.Count())
98              {
99
100                 if (words[i][1] == "Noun")
101                 {
102                     if (!isInside(i))
103                     {
104                         sum = float.Parse(words[i][3]);
105                         adjStr.Remove(0, adjStr.Length);
106
107                         j = 0; counter = 0;
108                         while (j < words.Count())
109                         {
110                             if (words[j][1] == "Adj" && words[
                                    j][2] == words[i][2])
111                             {
112                                 adjStr.Append(words[j][0] +
                                        "(");
113                                 if (float.Parse(words[j][3])
                                        >0.5)
114                                 {
115                                     adjStr.Append("положит");
```

```
116                         }
117                         else
118                             if (float.Parse(words[j
                                   ][3]) <-0.3)
119                         {
120                             adjStr.Append("негатив");
121                         }
122                         else
123                         {
124                             adjStr.Append("нейтрал");
125                         }
126
127                         adjStr.Append("");   ");
128
129                         sum += float.Parse(words[j
                               ][3]);
130                         counter++;
131
132                     }
133                     j++;
134                 }
135                 wordsTones.Add(words[i][0], sum/
                      counter);
136                  if (sum / counter > 0.5)
137                     {
138                         dataGrid.Rows.Add(words[i][0],
                              "положительная", adjStr.
                             ToString());
139
140                         numPOS++;
141                     }
142                  else if (sum / counter < -0.3)
143                     {
144                         dataGrid.Rows.Add(words[i][0],
                              "негативная" , adjStr.
                             ToString());
145
146                     numNEG++;
147                     }
148                   else
149                     {
150                         dataGrid.Rows.Add(words[i][0],
                              "нейтральная", adjStr.
                             ToString());
151
```

```
152                             numNEUT++;
153                             }
154                         }
155                     }
156                     i++;
157                 }
158 
159                 /*foreach (var w in wordsTones)
160                 {
161                     if (w.Value > 0.5)
162                     {
163                         dataGrid.Rows.Add(w.Key, "положительная");
164                         numPOS++;
165                     }
166                     else if (w.Value < -0.3)
167                     {
168                         dataGrid.Rows.Add(w.Key, "негативная");
169                         numNEG++;
170                     }
171                     else
172                     {
173                         dataGrid.Rows.Add(w.Key, "нейтральная");
174                         numNEUT++;
175                     }
176                 }*/
177 
178                 if (words.Count() == 0) MessageBox.Show("Атрибутов
                        не найдено");
179 
180                 numberPos.Text = numPOS.ToString();
181                 numberNeg.Text = numNEG.ToString();
182                 numberAll.Text = (numPOS + numNEG + numNEUT).
                        ToString();
183                 getAnswer(numPOS + numNEG + numNEUT, numPOS,
                        numNEG, numNEUT);
184 
185 
186             }
187 
188             private void deleteOutput()
189             {
190 
191                 if (File.Exists(@"C:/tomita/output.txt"))
192                 {
193                     File.Delete(@"C:/tomita/output.txt");
```

```
194             }
195         }
196         private void makeCodeTomita(int mode)
197         {
198             /*
199             * This void added code to first.cxx to prevent
                   errors like two adj loooks like noun + adj (ста
                   рший спортивный, мой рабочий)
200             */
201             string code;
202             if (mode == 0 || mode == 1)
203             {
204                 if (File.Exists(@"C:/tomita/first.cxx"))
205                 {
206                     File.Delete(@"C:/tomita/first.cxx");
207                 }
208 
209 
210                     if (mode == 1)
211                     {
212                         code = @"#encoding ""utf-8""    // coo
                               бщаем парсеру о том, в какой кодиро
                               вке написана грамматика
213 #GRAMMAR_ROOT S      // указываем корневой нетерминал граммати
    ки
214 
215 AdjCoord->Adj; //одно прилагатльное
216                 AdjCoord->AdjCoord <gnc-agr[1], rt> ','
                       AdjCoord <gnc-agr[1]>; // прилагательные -
                       однородные члены могут быть записаны через
                       запятую
217                 AdjCoord->AdjCoord <gnc-agr[1], rt> 'и'
                       AdjCoord <gnc-agr[1]>; // или через сочинит
                       ельный союз 'и'
218 
219 
220                 S->Adj interp(FactAdj.A::norm = ""m,sg""); ";
221 
222 
223                     }
224                     else
225                     {
226                         code = @"#encoding ""utf-8""    // coo
                               бщаем парсеру о том, в какой кодиро
                               вке написана грамматика
```

```
227 #GRAMMAR_ROOT S      // указываем корневой нетерминал граммати
    ки
228
229 AdjCoord->Adj; //одно прилагатльное
230                 AdjCoord->AdjCoord <gnc-agr[1], rt> ','
                       AdjCoord <gnc-agr[1]>; // прилагательные -
                       однородные члены могут быть записаны через
                       запятую
231                 AdjCoord->AdjCoord <gnc-agr[1], rt> 'и'
                       AdjCoord <gnc-agr[1]>; // или через сочинит
                       ельный союз 'и'
232
233                S->Noun<gnc-agr[1]> interp(Fact.Noun) AdjCoord<
                      gnc-agr[1]>+ interp(Fact.Adj::norm = ""m,sg
                      "");
234 S->AdjCoord <gnc-agr[1]> +interp(Fact.Adj::norm = ""m,sg"")
    Noun <gnc-agr[1]> interp(Fact.Noun); ";
235                     }
236                 //using (StreamWriter sw = new StreamWriter(@"
                       C:/tomita/first.cxx", false, System.Text.
                       Encoding.UTF8))
237                 //{
238                 //    sw.WriteLine(code);
239                 //}
240
241
242                 int NumberOfRetries = 3;
243                 int DelayOnRetry = 1000;
244
245                 for (int h = 1; h <= NumberOfRetries; h++)
246                 {
247                     try
248                     {
249
250                         using (StreamWriter sw = new
                               StreamWriter(@"C:/tomita/first.cxx
                               ", false, System.Text.Encoding.UTF8
                               ))
251                         {
252                             sw.WriteLine(code);
253                             sw.Close();
254                         }
255
256                         break;
257                     }
```

```
258                     catch (IOException e) when (h <=
                            NumberOfRetries)
259                     {
260                         Thread.Sleep(DelayOnRetry);
261                     }
262                 }
263
264
265             }
266
267             deleteOutput();
268         }
269
270         private bool isInside(int i)
271         {
272             return wordsTones.ContainsKey(words[i][0]);
273
274         }
275
276         private void GetText()
277         {
278             if (File.Exists(@"C:/tomita/test.txt"))
279             {
280                 File.Delete(@"C:/tomita/test.txt");
281             }
282             using (FileStream fstream = new FileStream("C:/
                    tomita/test.txt", FileMode.OpenOrCreate))
283             {
284                 byte[] array = System.Text.UTF8Encoding.UTF8.
                        GetBytes(textBox.Text);
285
286                 if (array.Length < 1000000000)
287                 {
288                     fstream.Write(array, 0, array.Length);
289                     Console.WriteLine("Текст записан в файл");
290                     fstream.Close();
291                 }
292                 else
293                 {
294                     MessageBox.Show("Введенный текст слишком в
                            елик.");
295                 }
296                 fstream.Close();
297             }
298
```

```
299         }
300
301         private void setAdjtext(string adjstr)
302         {
303             if (File.Exists(@"C:/tomita/test.txt"))
304             {
305                 File.Delete(@"C:/tomita/test.txt");
306             }
307             int NumberOfRetries = 3;
308             int DelayOnRetry = 1000;
309
310             for (int h = 1; h <= NumberOfRetries; h++)
311             {
312                 try
313                 {
314
315                     using (FileStream fstream = new FileStream
                           ("C:/tomita/test.txt", FileMode.
                           OpenOrCreate))
316                     {
317                         byte[] array = System.Text.
                               UTF8Encoding.UTF8.GetBytes(adjstr);
318                         fstream.Write(array, 0, array.Length);
319                         //Console.WriteLine("Текст записан в ф
                               айл");
320                         fstream.Close();
321                     }
322
323                     break;
324                 }
325                 catch (IOException e) when (h <=
                       NumberOfRetries)
326                 {
327                     Thread.Sleep(DelayOnRetry);
328                 }
329             }
330         }
331
332         private void StartTomita()
333         {
334             Process cmd = new Process();
335             cmd.StartInfo.FileName = "cmd.exe";
336             cmd.StartInfo.RedirectStandardInput = true;
337             cmd.StartInfo.RedirectStandardOutput = true;
338             cmd.StartInfo.CreateNoWindow = true;
```

```
339             cmd.StartInfo.UseShellExecute = false;
340             cmd.Start();
341             cmd.StandardInput.WriteLine(@"cd C:/tomita");
342             cmd.StandardInput.WriteLine(@"tomitaparser.exe
                    config.proto");
343             cmd.StandardInput.Flush();
344             cmd.StandardInput.Close();
345             cmd.Close();
346         }
347 
348         private void getTonalDictionary()
349         {
350             string temp;
351             StringBuilder word = new StringBuilder();
352             StringBuilder toneStr = new StringBuilder();
353             temp = File.ReadAllText(@"C:/tomita/tonalDic.txt",
                    Encoding.UTF8);
354             int i = 0;
355             while (i < temp.Length-10)
356             {
357 
358                 if (temp[i]=='\n')
359                 {
360                     word.Remove(0, word.Length);
361                     i++;
362                     while (temp[i] != ';')
363                     {
364                         word.Append(temp[i]);
365                         i++;
366                     }
367                     i++;
368                     while (temp[i] != ';') i++;
369                     toneStr.Remove(0, toneStr.Length);
370                     i++;
371                     while (temp[i] != ';')
372                     {
373                         toneStr.Append(temp[i]);
374                         i++;
375                     }
376                     tones.Add(word.ToString(), float.Parse(
                            toneStr.ToString()));
377                 }
378                 i++;
379             }
380 
```

```
381         }
382
383         private float getWorldTone(string word)
384         {
385
386             string str = word.Replace("e", "ë");
387             if (tones.ContainsKey(word))
388                 return tones.Where(p => p.Key == word).Select(
                       tones => tones.Value).First();
389             else
390             {
391                 if (word.Contains("e"))
392                 {
393                     if (tones.ContainsKey(str))
394                         return tones.Where(p => p.Key == str).
                               Select(tones => tones.Value).First
                               ();
395                 }
396
397                 return 0;
398             }
399
400         }
401
402         private void addWord(string word, float tone, bool
              isNoun) //adding the word to array words
403         {
404             int i = 0; bool isEnd = false;
405             // string temp="";
406             StringBuilder temp= new StringBuilder();
407             while (isEnd!=true)
408             {
409                 if (isNoun)
410                 {
411                     numEx = -1;
412                     for (int j =0; j < words.Count(); j++)
413                     {
414                         if (String.Equals(words[j][0], word))
415                         {
416                             numEx = int.Parse(words[j][2]);
417
418                         }
419                     }
420
421                     words.Add(new List<string>());
```

```
422
423                     words[numRows].Add(word);
424                     words[numRows].Add("Noun");
425                    if (numEx ==-1)  words[numRows].Add(numNoun
                         .ToString());
426                    else words[numRows].Add(numEx.ToString());
427                     words[numRows].Add(tone.ToString());
428                     numRows++;
429                     isEnd = true;
430                }
431                if (!isNoun)//if the word is Adj
432                {
433                     /*    1. Нужно разбить строку с прилагател
                           ьным
434                      *
435                      *    1.1. Поменять текст в test.txt
436                      *    1.2. Запусть парсер
437                      *    1.3. Получить факты из парсера и зан
                            ести каждое в words
438                      *
439                           2. Добавить в words  все полученные
                              прилагательные */
440
441                     if (word.Contains(" "))
442                     {
443                         setAdjtext(word);
444                         if (isFirstTime == true) { isFirstTime
                              = false; makeCodeTomita(1); }
445                         else { makeCodeTomita(2); }
446                         StartTomita();
447                         {
448                             temp.Clear();
449                             int NumberOfRetries = 3;
450                             int DelayOnRetry = 1000;
451                             for (int h = 1; h <=
                                   NumberOfRetries; h++)
452                             {
453                                 try
454                                 {
455                                     using (StreamReader sr =
                                           new StreamReader(@"C:/
                                           tomita/output.txt",
                                           Encoding.UTF8))
456                                     {
457
```

```
458                                 temp.Append(sr.
                                       ReadToEnd());
459                                 sr.Close();
460                             }
461                             break;
462                         }
463                         catch (IOException e) when (h
                               <= NumberOfRetries)
464                         {
465                             Thread.Sleep(DelayOnRetry)
                                   ;
466                         }
467                     }
468
469
470
471                 int num = 0;
472                 StringBuilder w = new StringBuilder();
473                 i = 0;
474                 while (num < temp.ToString().Length -
                       5)
475                 {
476                     if (temp.ToString()[num] == 'A' &&
                           temp.ToString()[num + 1] == '
                           ' && temp.ToString()[num + 2]
                           == '=')
477                     {
478                         num += 4;
479                         int j = num; i = 0;
480                         w.Remove(0, w.Length);
481                         while (temp.ToString()[j] !=
                               '\n')
482                         {
483                             w.Append(temp.ToString()[j
                                   ]);
484                             j++; i++;
485                         }
486                         words.Add(new List<string>());
487
488                         words[numRows].Add(w.ToString
                               ());
489                         words[numRows].Add("Adj");
490                         if (numEx==-1) words[numRows].
                               Add(numNoun.ToString());
```

```
491                                 else words[numRows].Add(numEx.
                                       ToString());
492                                 words[numRows].Add(
                                       getWorldTone(w.ToString()).
                                       ToString());
493                                 numRows++;
494                                 isEnd = true;
495                             }
496                             num++;
497                         }
498                         }
499                     }
500                     else
501                     {
502                         words.Add(new List<string>());
503                         words[numRows].Add(word.ToString());
504                         words[numRows].Add("Adj");
505                         if (numEx == -1) words[numRows].Add(
                               numNoun.ToString());
506                         else words[numRows].Add(numEx.ToString
                               ());
507                         words[numRows].Add(getWorldTone(word.
                               ToString()).ToString());
508                         numRows++;
509                         isEnd = true;
510                     }
511                 }
512
513             }
514         }
515
516         private void ReadFacts()
517         {
518                   string text="";
519                  int NumberOfRetries = 3;
520                  int DelayOnRetry = 1000;
521
522                 for (int h=1; h <= NumberOfRetries; h++) {
523                 try {
524
525                 using (StreamReader sr = new StreamReader(@"C
                       :/tomita/output.txt", Encoding.UTF8))
526                 {
527                     text = sr.ReadToEnd();
528                 }
```

```
529
530                 break;
531             }
532               catch (IOException e) when (h <= NumberOfRetries
                    )
533             {
534                 Thread.Sleep(DelayOnRetry);
535             }
536         }
537                 int num = 0;
538                 StringBuilder word = new StringBuilder();
539                 int i = 0;
540                 while (num < text.Length - 5)
541                 {
542                     if (text[num] == 'N' && text[num + 1] == '
                        o' && text[num + 2] == 'u' && text[num
                        + 3] == 'n' && text[num + 4] == ' ' &&
                        text[num + 5] == '=')
543                     {
544                         num += 7;
545                         int j = num; i = 0;
546                         word.Remove(0, word.Length);
547                         while (text[j] != '\n')
548                         {
549                             word.Append(text[j]);
550                             j++; i++;
551                         }
552                         num = j + 2;
553                         //функция которая находит в словаре сл
                            ово  и возвращает тональность
                            getWorldTone
554                         //вызов функции addWord(word, ton (
                            float), true)
555                         addWord(word.ToString(), getWorldTone(
                            word.ToString()), true);
556                         while (!(text[num] == 'A' && text[num
                            + 1] == 'd' && text[num + 2] == 'j'
                             && text[num + 3] == ' ' && text[
                            num + 4] == '='))
557                             num++;
558
559                         num += 6;
560                         i = 0; word.Remove(0, word.Length);
561                         while (text[num] != '\n')
562                         {
```

```
563                             word.Append(text[num]);
564                             num++; i++;
565                         }
566                         addWord(word.ToString(), getWorldTone(
                               word.ToString()), false);
567                         numNoun++;
568                     }
569 
570                     num++;
571                 }
572                 num++;
573             }
574         }
575 }
```

Листинг 1.2

Program.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Diagnostics;
4 using System.Linq;
5 using System.Threading.Tasks;
6 using System.Windows.Forms;
7 
8 namespace Tomita
9 {
10     static class Program
11     {
12         /// <summary>
13         /// The main entry point for the application.
14         /// </summary>
15         ///
16 
17         [STAThread]
18         static void Main()
19         {
20             Application.EnableVisualStyles();
21             Application.SetCompatibleTextRenderingDefault(
                   false);
22             Application.Run(new Form1());
23 
24         }
25     }
26 }
```

Листинг 1.3

facttypes.proto

```
import "base.proto";
import "facttypes_base.proto";
message Fact: NFactType.TFact
{
    required string Noun = 1;
    required string Adj = 2;
}
message FactAdj: NFactType.TFact
{
    required string A = 1;
}
```

Листинг 1.4

mydic.gzt

```
TAuxDicArticle "грамматика"
{
    key = { "tomita:first.cxx" type=CUSTOM }
}
```

Листинг 1.5

first.cxx

```
#encoding "utf-8"    // сообщаем парсеру о том, в какой кодиро
   вке написана грамматика
#GRAMMAR_ROOT S      // указываем корневой нетерминал граммати
   ки

AdjCoord->Adj; //одно прилагатльное
AdjCoord->AdjCoord < gnc - agr[1], rt> ',' AdjCoord <gnc - agr
   [1]>; // прилагательные - однородные члены могут быть запис
   аны через запятую
AdjCoord->AdjCoord < gnc - agr[1], rt> 'и' AdjCoord <gnc - agr
   [1]>; // или через сочинительный союз 'и'

S->Noun<gnc - agr[1]> interp(Fact.Noun) AdjCoord<gnc - agr
   [1]>+interp(Fact.Adj::norm = "m, sg");
S->AdjCoord <gnc - agr[1]> +interp(Fact.Adj::norm = "m, sg")
   Noun <gnc - agr[1]> interp(Fact.Noun);
```

Листинг 1.6

config.proto

```
encoding "utf8"; // указываем кодировку, в которой написан кон
   фигурационный файл
```

```
2
3  TTextMinerConfig {
4    Dictionary = "mydic.gzt"; // путь к корневому словарю
5
6    PrettyOutput = "PrettyOutput.html"; // путь к файлу с отладо
       чным выводом в удобном для чтения виде
7
8    Input = {
9      File = "test.txt"; // путь к входному файлу
10   }
11   Output = {File = "output.txt"
12               Format = text}
13 Facts = [
14     { Name = "Fact" },
15     { Name = "FactAdj" }
16 ]
17   Articles = [
18     { Name = "грамматика" } // название статьи в корневом слов
         ape,
19                                              // которая содержит
                                                   запускаемую грамм
                                                   атику
20   ]
21 }
```