ՀՀ ԳԻՏՈՒԹՅՈՒՆՆԵՐԻ ԱԶԳԱՅԻՆ ԱԿԱԴԵՄԻԱՅԻ
ԻՆՖՈՐՄԱՏԻԿԱՅԻ և ԱՎՏՈՄԱՏԱՑՄԱՆ ՊՐՈԲԼԵՄՆԵՐԻ ԻՆՍՏԻՏՈՒՏ

Լևոն Ռոբերտի Հակոբյան

UNL ԼԵԶՎԻՑ ԴԵՊԻ ԲՆԱԿԱՆ ԼԵԶՈՒ ՓՈԽԱՐԿՄԱՆ ԿԱՆՈՆՆԵՐԻ ՀԱՎԱՔԱԾՈՒՆԵՐԻ ԸՆԴԼԱՅՆՄԱՆ ԵՎ ՈՒՂՂՄԱՆ ԱՎՏՈՄԱՏԱՑՎԱԾ ՀԱՄԱԿԱՐԳ

Ե.13.04 – «Հաշվողական մեքենաների, համալիրների, համակարգերի և ցանցերի մաթեմատիկական և ծրագրային ապահովում»
տեխնիկական գիտությունների թեկնածուի
գիտական աստիճանի հայցնամ ատենախոսության

ՍԵՂՄԱԳԻՐ

ԵՐԵՎԱՆ 2014

**ИНСТИТУТ ПРОБЛЕМ ИНФОРМАТИКИ И АВТОМАТИЗАЦИИ
НАЦИОНАЛЬНОЙ АКАДЕМИИ НАУК РА**

Акопян Левон Робертович

АВТОМАТИЗИРОВАННАЯ СИСТЕМА ДЛЯ РАСШИРЕНИЯ И КОРРЕКТИРОВАНИЯ НАБОРОВ ПРАВИЛ ТРАНСФОРМАЦИИ С ЯЗЫКА UNL НА ЕСТЕСТВЕННЫЙ ЯЗЫК

АВТОРЕФЕРАТ

диссертации на соискание ученой степени кандидата
технических наук по специальности
05.13.04 – «Математическое и программное обеспечение вычислительных машин, комплексов, систем и сетей»

ЕРЕВАН 2014

Ատենախոսության թեման հաստատվել է ՀՀ ԳԱԱ ինֆորմատիկայի և ավտոմատացման պրոբլեմների ինստիտուտում:

Գիտական ղեկավար՝ ֆիզ.մաթ.գիտ.դոկտոր Ի.Դ. Զասլավսկի

Պաշտոնական ընդդիմախոսներ՝ ֆիզ.մաթ.գիտ.դոկտոր Է.Մ.Պողոսյան
տեխ.գիտ.թեկնածու Վ.Ա.Ավետիսյան

Առաջատար կազմակերպություն՝ Երևանի պետական համալսարան

Պաշտպանությունը տեղի կունենա 2014թ. դեկտեմբերի 12-ին ժ. 16:00-ին ՀՀ ԳԱԱ Ինֆորմատիկայի և ավտոմատացման պրոպլեմների ինստիտուտում գործող 037 «Ինֆորմատիկա և հաշվողական համակարգեր» մասնագիտական խորհրդի նիստում, հետևյալ հասցեով՝ 0014 , Երևան, Պ. Սևակի 1:

Ատենախոսությանը կարելի է ծանոթանալ ՀՀ ԳԱԱ ԻԱՊԻ գրադարանում:
Սեղմագիրն առաքված է 2014թ. նոյեմբերի 12-ին:

Մասնագիտական խորհրդի գիտական
քարտուղար, ֆ.մ.գ.դ. Հ.Գ.Սարուխանյան

Тема диссертации утверждена в Институте проблем информатики и автоматизации НАН РА

Научный руководитель: доктор физ.мат.наук И.Д. Заславский

Официальные оппоненты: доктор физ.мат.наук Э.М.Погосян
кандидат тех.наук В.А.Аветисян

Ведущая организация: Ереванский государственный университет

Защита состоится 12 декабря 2014г. в 16:00 на заседании специализированного совета 037 “Информатика и вычислительные системы” Института проблем информатики и автоматизации НАН РА по адресу: 0014, г. Ереван, ул. Паруйра Севака, 1.

С диссертацией можно ознакомиться в библиотеке ИПИА НАН РА.
Автореферат разослан 12-го ноября 2014г.

Ученый секретарь специализированного
совета, д.ф.м.н. А.Г.Саруханян

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Интенсивное развитие компьютерных технологий и их широкое использование в повседневной жизни повлекло за собой спрос на технологии, повышающие эффективность коммуникации между пользователями. В связи с этим актуальными стали проблемы машинного перевода, семантического анализа текста, а также другие аналогичные проблемы.[1]

В 1996-ом году в Институте Исследований Университета Объединенных Наций (Institute of Advanced Studies) в Токио был начат проект по созданию искусственного языка, называющегося "Универсальным сетевым языком" (Universal Networking Language, сокращенно UNL), который позволяет решать проблемы как автоматизированного перевода (в рамках UNL называемого также трансформацией) с одного естественного языка на другой, так и предоставляет возможность машинной обработки семантической информации.[2]

В 2001 году к проекту присоединилась Армения, и начались работы в рамках ИПИА НАН РА под руководством Юрия Шукуряна и Владимира Саакяна.

Отметим, что язык UNL является искусственным языком для представления и обработки знаний и информации, описанной на различных естественных языках. Язык UNL также используется в качестве языка-посредника при автоматизированных переводах с одного естественного языка на другой.

Предложение на языке UNL имеет вид ориентированного графа или орграфа (семантический граф, semantic network, UNL graph), в котором вершины соответствуют семантическим единицам языка, т. е. словам и словоформам (универсальные слова, Universal Words, UW, concepts), а дуги (универсальные отношения, Universal Relations) соответствуют семантическим связям и отношениям между семантическими единицами.

Трансформация с языка UNL на естественный язык, а также обратная трансформация происходит при помощи UNL-грамматики и словарей. UNL-грамматика содержит правила, по которым происходит трансформация, а словари содержат записи об универсальных словах, их переводах и о свойствах соответствующих им атрибутов.

Сегодня актуальным является вопрос создания и развития наборов правил, позволяющих производить трансформацию с естественного языка на язык UNL и обратно. Важной составляющей процесса создания наборов правил является наличие соответствующих программных инструментов, позволяющих эффективно корректировать имеющиеся наборы правил. Кроме того, существенным является создание теоретического аппарата, позволяющего формализовать процесс корректировки наборов правил. В работе описывается один из методов решения этой задачи. А именно, дается метод корректирования и расширения наборов правил трансформации с языка

---

[1] И. А. Мельчук, Опыт теории лингвистических моделей 'Смысл<->Текст' Москва, Наука, 1974.

[2] H. Uchida, M. Zhu, and T. Della Senta, A Gift for a Millennium, IAS/UNU, 1999.
UNDL Foundation web site, [Online]. Available: http://www.undlfoundation.org/

UNL на естественный язык. Этот метод позволяет производить отладку наборов правил в автоматизированном режиме.

Отметим, что подобные исследования проводились и в Армении начиная с 1957 года в лаборатории математической логики под руководством Игоря Заславского и в рамках проекта Русско-Армянского машинного переводчика под руководством Теодора Тер-Микаеляна, Владимира Григоряна и Роберта Урутяна. Сегодня исследования в этой области проводятся также в ИПИА НАН РА под руководством Эдуарда Погосяна[3].

Актуальность работы

Так как эффективность использования UNL в качестве языка-посредника для автоматических переводов зависит от используемых ресурсов (словарь, набор правил), специалистам, работающим над созданием, расширением и тестированиям подобных ресурсов существенно иметь соответствующие программные инструменты, позволяющие выявлять некорректные правила и словарные записи, приводящие к ошибкам, и, по возможности, указывать пути устранения этих ошибок.

В работе представлена программная система, позволяющая решать подобные задачи, в частности, производить отладку наборов правил в автоматизированном режиме.

Для решения подобных задач разработаны алгоритмы и программы, позволяющие пользователю находить правила и словарные записи, приведшие к той или иной ошибке, а, в ряде случаев, и получать рекомендации относительно возможных путей устранения ошибки.

Существенное препятствие в процессе отладки правил состоит в трудности эффективного получения наборов альтернативных путей трансформации и выявления среди них наиболее приемлемых. Пользователю для этого приходилось вручную выбирать правила для применения на данном шаге трансформации и запоминать информацию о полученных вариантах.

Эта задача решается при помощи алгоритмов и программ для построения так называемого «дерева путей трансформации». Соответствующий алгоритм позволяет пользователю в автоматическом режиме получить все существенные пути трансформации и их результаты. Алгоритм также дает возможность сравнения полученных результатов с эталоном, заданным пользователем, при помощи алгоритмов оценки качества машинного перевода.

Цель диссертационной работы

Целью диссертационной работы является разработка алгоритмов и программ, позволяющих эффективно расширять и корректировать рассматриваемые наборы правил трансформации и словарные записи. Разработанные алгоритмы и программы дают возможность нахождения некорректных правил и словарных записей и предлагают пользователю возможные способы их корректирования.

[3] Edward Pogossian, "On adequate and constructive models of processing of meanings", Computer Science and Information Technologies (CSIT) 2013, pp. 1-12

Научная новизна

1. Разработаны новые алгоритмы и программы, позволяющие получать так называемое дерево возможных путей трансформации, что позволяет эффективно получать наборы различных возможных путей трансформации.

2. Дана классификация ошибок, возникающих при работе алгоритма трансформации, что позволяет указывать свойства возникшей ошибки, а также применять алгоритмы корректирования правил трансформации в соответствии с классами ошибок.

3. Созданы алгоритмы и программы, позволяющие выявлять правила, повлиявшие на указанный пользователем узел в процессе трансформации, и алгоритмы и программы для корректирования соответствующих типов ошибок, которые находят ответственные за данную ошибку правила и словарные записи.

Используемые технологии

Алгоритмы и программы, созданные в рамках данной диссертационной работы, интегрированы в систему UNDL Platform, поэтому при их реализации были использованы технологии, имеющиеся в данной системе. Исходя из этого языком программирования был выбран язык Java 6.0. Так как система UNDL Platform является веб-приложением, то есть создана для работы в среде Интернет, в ходе работы были также использованы технологии сетевого программирования, такие как: Java Servlet, Java Server Pages, JavaScript, CSS2, HTML4 и технология AJaX.[4]

Практическая ценность

Использование специалистами автоматизированного метода корректирования ошибок, возникающих в ходе трансформации, позволяет усовершенствовать процесс разработки наборов правил. Это позволит улучшить качество переводов с использованием UNL.

Основные положения выносимые на защиту

1. Разработаны алгоритмы и программы построения дерева возможных путей трансформации; методы их применения в комбинированном варианте и в случае построения поддеревьев, а также методы сравнения результатов (листьев), полученных в ходе построения деревьев.

2. Разработаны алгоритмы и программы корректирования возможных ошибок в результате трансформации, среди которых имеются алгоритмы и программы нахождения повлиявших на данный узел правил и алгоритмы и программы корректирования ошибок соответствующих типов.

3. Разработаны методы хранения примеров применений правил трансформации и использования этой информации в процессе дальнейшей корректировки наборов правил.

4. Разработана программная среда пользователя, позволяющая комбинировать применение вышеперечисленных алгоритмов и программ, с целью наиболее эффективного их использования.

---

[4] The UNL Community Portal web site, [Online]. Available: http://www.unlweb.net/unlweb/

Внедрение результатов работы

Программная реализация описанных алгоритмов использована в работе UNL-центра Армении, в рамках проекта по разработке словарей и наборов правил трансформации.

Апробация диссертационной работы

Результаты, полученные в работе, доложены на следующих конференциях и семинарах: Седьмая годичная научная конференция РАУ, 2012; Конференция The 23rd Meeting of Computational Linguistics In the Netherlands, 2013; Научный семинар ИПИА НАН РА, 2014;

Публикации

Материалы диссертационной работы напечатаны в 3 публикациях, в том числе в материалах одной международной конференции, и в двух статьях, список которых приведен в конце автореферата.

Структура диссертации

Диссертационная работа состоит из введения, пяти глав, заключения, списка литературы из 35 наименований. Общий объем диссертации составляет 105 страниц машинописного текста, включая 17 рисунков и 3 таблицы. Диссертация написана на русском языке.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

Во введении обоснована актуальность темы исследования, обозначена цель, основные задачи и методы их решения.

В первой главе дан обзор современных исследований в области использования языка UNL в качестве языка-посредника для автоматизированных переводов. Также в этой главе рассмотрена существующая в данной области проблематика.

В этой главе описываются также основные особенности языка UNL. Дается описание синтаксиса языка UNL; лингвистических ресурсов, используемых в процессе перевода с одного естественного языка на другой, таких как словари и наборы правил; рассматривается структура документов на основе UNL.

В главе также дан краткий обзор системы UNDL Platform. Данная система является на сегодняшний день набором основных инструментов для работы с UNL. В рамках проекта UNDL Platform реализованы алгоритмы и программы, описанные в работе.

Указывается также ряд введенных усовершенствований и возможных направлений развития системы UNDL Platform для обеспечения лучшего взаимодействия пользователя с системой при создании и расширении наборов правил и словарей.

Во второй главе дается описание алгоритмов**,** позволяющих получить все существенные варианты трансформации из языка UNL в естественный язык при разных последовательностях применения правил. Дается описание дерева возможных путей трансформации. Приведены методы построения указанного дерева и проводится сравнительный анализ этих методов. Также в этой главе указаны методы для обработки результатов, полученных при построении дерева возможных путей трансформации.

В первой части этой главы описан алгоритм трансформации, существующий в системе UNDL Platform. Трансформация с языка UNL на естественный язык осуществляется алгоритмом (именуемым «генератором»), который получает на вход предложение на языке UNL, словарь и набор соответствующих правил трансформации. Результатом работы генератора является предложение на естественном языке.

Рассматриваемые правила трансформации делятся на две группы: модифицирующие и определяющие правила. Модифицирующие правила - это пары типа (A, B), где A - условие, B - описание действий. Если какая-то часть предложения удовлетворяет условию A, применяется данное модифицирующие правило путем модификации предложения так, как это описано в B. Определяющие правила - это пары типа (C, D), где C - условие, D – целое число в промежутке от 0 до 255. Определяющее правило фактически показывает приоритетность появления ситуации, описанной в условии C.

В новейших разработках в сфере UNL существенно используется один подкласс определяющих правил - это так называемые «запрещающие правила». Запрещающим правилом называется определяющие правило, приоритет которого равен 0.

Принцип работы алгоритма трансформации можно изложить следующим образом. Алгоритм строится на основе так называемого «внешнего цикла», Каждый из шагов внешнего цикла реализуется при помощи нескольких так называемых «внутренних» циклов, описанных ниже.

Работа внешнего цикла состоит в проведении последовательных шагов процесса трансформации; в результате получается последовательность промежуточных конструкций. Работа внешнего цикла заканчивается после получения окончательного результата в виде предложения на естественном языке.

Каждый шаг работы внешнего цикла проводится посредством применения шагов так называемого «первого внутреннего цикла». При этом каждый шаг работы «первого внутреннего цикла» производится посредством применения шагов так называемого «второго внутреннего цикла». Таким образом, второй внутренний цикл как бы «вложен» в первый.

Перед каждым шагом внешнего цикла производится копирование текущей промежуточной конструкции. Эта копия служит для проверки правил, являющихся кандидатами на применение на текущем шаге.

При работе первого внутреннего цикла поочередно проверяются все правила в той очередности, в которой они хранятся в наборе правил. Как только находится правило, применимое к данной промежуточной конструкции, работа цикла временно останавливается и рассматривается вопрос о возможности применения найденного модифицирующего правила к рассматриваемой промежуточной конструкции.

Для выяснения такой возможности данное модифицирующее правило применяется на копии оригинальной промежуточной конструкции, и работу начинает второй внутренний цикл, проверяющий определяющие правила в той очередности, в которой они расположены в наборе правил.

Если все определяющие правила просмотрены, и среди них нет ни одного правила, применимого к данной промежуточной конструкции, то происходит применение соответствующего модифицирующего правила к текущей промежуточной конструкции и переход к следующему шагу внешнего цикла.

Если находится запрещающее правило, условию которого удовлетворяет рассматриваемая промежуточная конструкция, то применение модифицирующего правила отменяется, и происходит переход к следующему шагу первого внутреннего цикла для нахождения следующего модифицирующего правила, подходящего для применения.

Если находится определяющее, но не запрещающее правило, условию которого удовлетворяет рассматриваемая промежуточная конструкция, то данному модифицирующему правилу присваивается временный приоритет, равный приоритету, указанному в найденном определяющем правиле. Далее, в цикле проверяются все остальные модифицирующие правила, и тем из них, которые подходят к применению к рассматриваемой промежуточной конструкции, таким же образом присваивается приоритет. Если определяющих правил для текущего модифицирующего правила нет, то ему временно присваивается приоритет равный 0,5.

После просмотра всех модифицирующих правил, среди них (в основном варианте алгоритма) выбирается самое приоритетное правило, после чего происходит применение этого правила на оригинальном предложении и переход к следующему шагу внешнего цикла. Наряду с указанным основным вариантом алгоритма трансформации в работе применяется расширенные варианты этого алгоритма.

Далее в работе приводятся описания расширенных вариантов основного алгоритма.

В случае, если есть два и более модифицирующих правила с разными приоритетами, применимыми к текущей конструкции, учет этого обстоятельства производится различным образом в двух разновидностях расширенного алгоритма трансформации, а именно: во-первых, алгоритм построения полного дерева путей трансформации, во-вторых, алгоритм построения ограниченного дерева путей трансформации. Рассмотрим кратко эти алгоритмы. О структуре дерева путей трансформации будет сказано подробнее в следующем разделе.

Алгоритм построения полного дерева путей трансформации строится на основе учета всех модифицирующих правил, применимых к текущей промежуточной конструкции. В случае построения полного дерева путей трансформации приоритеты не рассматриваются.

Для применения рассматриваемого алгоритма предварительно для каждого случая, когда возможны разветвления путей трансформации, составляется список возможных направлений путей трансформации (каждое такое направление соответствует применяемому модифицирующему правилу); затем эти направления нумеруются натуральными числами. Каждый путь трансформации тогда однозначно определяется набором натуральных чисел, определяющих направления путей трансформации для каждого случая разветвления этих путей. Эти наборы упорядочиваются в словарном

(лексикографическом) порядке. Применение алгоритма трансформации проводится поочередно для каждого из упомянутых наборов натуральных чисел.

Для каждого фиксированного набора натуральных чисел и для каждой промежуточной конструкции происходит применение соответствующего модифицирующего правила, после чего происходит переход к следующему шагу внешнего цикла. Построение полного дерева путей трансформации заканчиваются, когда указанная процедура оказывается примененной для всех наборов натуральных чисел, упомянутых выше.

Нужно отметить, что построение полного дерева путей трансформации требует в большинстве случаев значительных вычислительных ресурсов, часто оно бывает неэффективным.

Алгоритм построения ограниченного дерева путей трансформации работает по тому же принципу, что и алгоритм построения полного дерева этих путей, с той лишь разницей, что в этом алгоритме используются не все разветвления путей трансформации, а только некоторые из них. Выбор используемых разветвлений путей трансформации в этом случае производится посредством применения особых алгоритмов («метод поиска в глубину», «метод поиска в ширину»); эти алгоритмы рассматриваются в нижеследующих разделах.

Как было сказано выше, при игнорировании приоритетов возможно получить более одной последовательности применимых правил, и на основе этого можно построить так называемое дерево возможных путей трансформации предложения.

Вершиной в данном дереве служит структура, содержащая, кроме прочих вспомогательных данных, UNL предложение или промежуточную конструкцию и правило, применением которого данная конструкция получена. Нам потребуется рассмотреть структуру этого дерева подробнее.

Каждая дочерняя вершина в дереве, кроме прочих вспомогательных данных, содержит информацию о соответствующем правиле и промежуточной конструкции, полученной посредством применения этого правила к промежуточной конструкции родительской вершины.

Соответственно, правило трансформации в корневой вершине будет пустым, а предложением будет исходное предложение на языке UNL, заданное для трансформации. Листья, полученные после построения дерева, будут содержать всевозможные варианты трансформации, согласованные со структурой соответствующего алгоритма, о котором пойдет речь далее.

Определение 2.1. Говорим, что *правило применимо* к предложению на языке UNL или к данной промежуточной конструкции, если в данном предложении или промежуточной конструкции есть подграф, удовлетворяющий условию правила.

Определение 2.2. *Применением правила* называется преобразование предложения при помощи модифицирующего правила, обозначаемое как $r : s \vdash s'$, при условии, что r является применимым к s.

Определение 2.3. Вершиной в дереве возможных путей трансформации является кортеж $v = (s,\ r,\ info)$, где s - UNL предложение или промежуточная конструкция, r - правило трансформации, $info$ – структура, содержащая дополнительную информацию и полученная посредством применения правила r к конструкции s.

Через V обозначим множество всевозможных вершин дерева возможных путей трансформации.

Определение 2.5. Пусть $v \in V$. *Дочерней* по отношению к v называется такая вершина v', которая может быть получена из v при помощи применения какого-либо модифицирующего правила.

Отношение между родительской вершиной v и дочерней вершиной v' обозначим как $v \vdash v'$. Посредством $D(v)$ обозначаем множество вершин, дочерних по отношению к вершине v. То есть, $D(v) = \{v_i \mid v \vdash v_i\}$.

Определение 2.7. Пусть $v_1', v_2' \in V$. Вершина v_1' называется *достижимой* из вершины v_2', если $\exists\ v_1, \dots, v_m \in V$, такие, что $v_1 \vdash v_2 \vdash \cdots \vdash v_{m-1} \vdash v_m$, $v_1' = v_1$, $v_2' = v_m$. Обозначим это обстоятельство посредством $v_1' \vdash^* v_2'$

Посредством $AV(v')$ обозначим множество вершин, достижимых из v'. То есть $AV(v') = \{v_i \mid v' \vdash^* v_i\}$.

Исходя из сказанного выше, можно определить дерево возможных путей трансформации следующим образом.

Определение 2.8. Полное дерево возможных путей трансформации – это ацикличный граф, в котором множество вершин есть $\{v_0, v_1, \dots, v_n\}$, где v_0 – корневая вершина, а $\{v_1, \dots, v_n\}$ – это все вершины, достижимые из корневой вершины, то есть $v_0 \vdash^* v_i$, а множество дуг - это множество пар $\{(v_{j_1}', v_{j_2}')\}$, где v_{j_1} – вершина графа, а $v_{j_2} \in D(v_{j_1})$.

Определение 2.9. *Изначальный путь трансформации, исходящий из данной вершины дерева* (или просто «*изначальный путь, исходящий из данной вершины*») – это такой путь трансформации, исходящий из данной вершины, когда при каждом разветвлении путей трансформации, описываемом соответствующим набором натуральных чисел (c_1, c_2, ... , c_t), где $c_1 < c_2 < \cdots < c_t$ (эти наборы рассмотрены выше) в качестве указателя дальнейшего пути трансформации берется наименьшее число c_1.

Если рассматривается изначальный путь трансформации, исходящий из корневой вершины дерева, то такой путь трансформации мы будем называть «изначальным путем трансформации» или даже просто «изначальным путем».

Далее в главе описываются методы, позволяющие сократить время работы алгоритма. При построении соответствующего интерфейса для доступа к вершинам в соответствии с их глубиной в дереве используется так называемая хеш-таблица. Хеш-таблица обеспечивает быстрый доступ к вершинам и позволяет отсекать повторяющиеся пути в дереве возможных путей трансформации.

Первый подход к построению дерева возможных направлений трансформации основан на методе поиск в глубину в дереве. Алгоритм состоит из внешнего цикла, каждая итерация которого состоит в обработке информации, относящейся к текущей вершине.

В первой итерации в качестве текущей вершины рассматривается корневая вершина. Итерация заключается в проведении следующих действий. Для текущей вершины запускается обычный процесс трансформации, проходящий изначальный путь, исходящий из этой вершины. В случае, когда текущей является корневая вершина, это позволяет пройти изначальный путь в дереве от корня до одного из листьев. После завершения процесса трансформации получаем соответствующий лист данного дерева, а также информацию об изначальном пути, на котором этот лист получен.

Дальнейшие шаги алгоритма поиска в глубину проводятся в соответствии со следующими определениями.

Определение 2.10. *Регулярным поддеревом* данного дерева будем называть поддерево данного дерева, обладающее следующими свойствами: (а) корневая вершина регулярного поддерева совпадает с корневой вершиной данного дерева; (б) для всякой вершины V регулярного поддерева данного дерева существует путь, ведущий из корневой вершины в вершину V и содержащийся в рассматриваемом поддереве.

Очевидно, что изначальный путь трансформации, ведущий от корневой вершины к какому-либо из листьев данного дерева, можно рассматривать как регулярное поддерево данного дерева.

Определение 2.11. *Висячей вершиной* регулярного поддерева F данного дерева G будем называть всякий лист поддерева F, не являющийся листом данного дерева G.

Работа алгоритм поиска в глубину заключается в построении последовательности поддеревьев $F_1,\ F_2,$ ... данного дерева G; эта последовательность получается в результате работы «внешнего цикла» следующим образом.

В качестве исходного дерева F_1 рассматривается упомянутый выше изначальный путь, ведущий от корневой вершины к одному из листьев дерева G. Далее, если уже построено поддерево F_i дерева G, выбирается одна из висячих вершин поддерева F_i, и при помощи такой же процедуры, как выше, строится путь от этой висячей вершины к одному и листьев дерева G; поддерево F_{i+1} получается из поддерева F_i посредством присоединения построенного пути к поддереву F_i.

Полученный в результате лист сохраняется в структуре данных, содержащей как полученный лист, так и соответствующую последовательность идентификаторов тех правил, которые описывают путь от корня к указанному листу. Все полученные в ходе работы алгоритма листья дерева сохраняются в едином списке.

Рассмотренные висячие вершины сохраняются в системе при помощи объектов специального класса. Объекты данного класса, кроме прочих данных, содержат последовательность правил, при помощи которых данная висячая вершина была получена из корневой вершины. Ссылки на висячие вершины добавляются в упомянутый выше список.

Далее, работает внешний цикл алгоритма, на каждом шаге которого выбирается следующая висячая вершина. Эта вершина принимается в качестве текущей вершины на данной итерации алгоритма. Внешний цикл повторяется до тех пор, пока в поддеревьях F_i есть висячие вершины. То есть до тех пор, пока не просмотрены все элементы

упомянутого выше списка. Таким образом, после завершения внешнего цикла получается полное дерево возможных путей трансформации. Все полученные листья будут сохранены в соответствующем списке.

При построении ограниченного дерева путей трансформации описанный выше алгоритм работает не до конца. Он сочетается, в частности, с алгоритмом поиска «в ширину».

Второй метод построения дерева возможных путей трансформации основан на поиске в ширину. Данный метод реализован следующим алгоритмом. Алгоритм состоит из двух основных циклов: внешнего и внутреннего. Внешний цикл обеспечивает обход уровней в дереве, а внутренний - обход вершин на текущем уровне. При этом уровнем в дереве называем множество вершин данного дерева, имеющих одинаковую глубину.

На каждом шаге внешнего цикла внутренний цикл обеспечивает обход вершин на текущем уровне и при этом происходит построение дочерних вершин для каждой из этих вершин. В указанном алгоритме используется модифицированная структура вершин дерева, которая, кроме прочих данных, содержит список указателей на дочерние вершины, а также атрибут, указывающий идентичную вершину. Этот атрибут будет рассмотрен ниже. Дочерние вершины генерируются специальным алгоритмом пошаговой трансформации, который трансформирует предложение применением только одного правила; оно передается ей в качестве параметра. Из множества правил выбираются те правила, которые применимы к промежуточной конструкции, соответствующей текущей вершине. Для каждого из этих правил вызывается алгоритм пошаговой трансформации, которому передается информация о данном правиле. Алгоритм возвращает новую промежуточную конструкцию, для которой строится новая вершина в дереве. Ссылка на эту вершину добавляется в соответствующий атрибут родительской вершины. Таким образом, в соответствующем списке родительской вершины добавляются ссылки на все дочерние вершины. Внешний алгоритм работает до тех пор, пока на текущем уровне возможна генерация дочерних вершин.

Описанная выше базовая часть алгоритма дополняется возможностью сравнения вершин одного уровня для исключения ситуаций, когда расчеты для сходных промежуточных конструкций вершин производятся дважды. Начиная с вершин, имеющих глубину три и более, в конце каждого шага внешнего цикла, то есть после построения всех дочерних вершин текущего уровня, вызывается алгоритм проверки вершин на идентичность. Данный алгоритм использует механизм быстрого доступа к вершинам по индексу, описанный выше, для доступа к текущим дочерним вершинам и их группировки. Группировка вершин производится по критерию схожести промежуточных конструкций, содержащихся в данных вершинах. Атрибуту, указывающему идентичную вершину для всех вершин группы, кроме первой вершины, присваивается значение ссылки на первую вершину в группе. Таким образом, только одна вершина в каждой группе будет содержать пустое значение атрибута, указывающего идентичную вершину. Далее построение дерева продолжается только для вершин, имеющих пустое значение атрибута, указывающего на идентичную вершину.

Описанное выше усовершенствование позволяет после получения вершин данного уровня сгруппировать подобные вершины и продолжать поиск в дереве для единственной вершины из группы. Тем самым можно отсечь пути трансформации, ведущие к повторяющимся результатам и сократить время вычислений.

При реализации алгоритма поиск в ширину ведется до определенной глубины в дереве, а после достижения данной глубины осуществляется поиск в глубину для соответствующих вершин.

Построение полного дерева возможных путей трансформации требует большого количества вычислительных ресурсов. При работе специалистов в области лингвистики часто возникают ситуации, когда нет необходимости в построении полного дерева, однако требуется получить некоторые промежуточное конструкции. В работе описан метод, позволяющий пользователю получить ограниченные деревья путей трансформации.

В результате работы алгоритма построения дерева возможных путей трансформации получается большое количество листьев. При этом возникает неудобства для пользователей с обработкой данной информации. В главе описаны методы обработки полученных результатов с использованием таких алгоритмов оценки качества машинных переводов, как BLEU и METEOR[5].

Оба упомянутых алгоритма требуют наличия эталонных примеров для сравнения результата перевода. Следовательно, пользователю следует указать эталон перевода рассматриваемого UNL предложения на естественном языке.

В системе UNDL Platform был реализован также алгоритм сравнения листьев, который позволяет сравнивать листья, используя указанные выше алгоритмы. Алгоритм сравнивает эталон со всеми листьями, а точнее с результатами трансформации, содержащимися в листьях, и упорядочивает листья в новой последовательности в соответствии с их близостью к эталону. Это позволяет пользователю получить наиболее близкие к эталону результаты и оценить их удаленность от эталона. На основе этого пользователь может сделать вывод о наличии либо отсутствии в дереве возможных путей трансформации удовлетворяющего пути.

В третьей главе описывается метод корректировки возможных ошибок, возникающих при работе алгоритма трансформации из языка UNL в естественный язык.

Метод работает в полуавтоматическом режиме, а именно: ошибки в работе алгоритма трансформации указываются пользователем, после чего алгоритм корректировки автоматически находит правило трансформации (возможно, группу правил), ответственное за появление ошибки и выдает рекомендации по изменению соответствующего правила (возможно, нескольких правил).

---

[5] Banerjee, S. and Lavie, A. (2005) "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments" in *Proceedings of Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization at the 43rd Annual Meeting of the Association of Computational Linguistics (ACL-2005), Ann Arbor, Michigan, June 2005*

Далее в главе приводится постановка задачи, которая может быть охарактеризована следующим образом: дать метод, позволяющий по данной ошибке указать те правила, применение которых могло повлечь за собой возникновение данной ошибки, а также выдать рекомендации по корректированию этих правил. Такой метод, в частности, основывается на соответствующей классификации ошибок. Рассмотрим классификацию ошибок, применяемую в данной работе.

Отметим, что в результате трансформации, в частности, получается линейная структура, состоящая из N вершин: $a_1\ a_2 \dots a_N$. При этом, каждая из вершин a_i задается последовательностью символов: $a_i = l_1^{(i)} \dots l_{M_i}^{(i)}$.

В рамках данной структуры можно указать следующие типы возможных ошибок:

1. Неправильная расстановка вершин: в последовательности $a_1 \dots a_i \dots a_j \dots a_N$ возможна ситуация, когда вершины a_i и a_j следует поменять местами для корректной расстановки вершин в предложении.

2. Отсутствие необходимой вершины: в последовательности $a_1 \dots a_i a_{i+1} \dots a_N$ возможна ситуация, когда при корректном переводе между вершинами a_i и a_{i+1} должна находится какая-то вершина b.

3. Лишняя вершина: в последовательности $a_1 \dots a_i \dots a_N$ возможна ситуация, когда вершина a_i является лишней вершиной в предложении.

4. Ошибка в вершине: допустим, что в имеющейся последовательности $a_1 \dots a_i \dots a_N$, одна из вершин $a_i = l_1^{(i)} \dots l_j^{(i)} \dots l_{j+t}^{(i)} \dots l_{M_i}^{(i)}$, содержит ошибку. Тогда ошибка содержится в некотором промежутке $l_j^{(i)} \dots l_{j+t}^{(i)}$. В этом случае пользователю надлежит также указать промежуток символов, в котором содержится ошибка. Если ошибок в данной вершине несколько, то эти случаи рассматриваются как отдельные ошибки.

5. Прочие виды ошибок в рамках данной работы не рассматриваются.

Как было сказано выше, процесс трансформации предложения на языке UNL в предложение естественного языка состоит в последовательном применении модифицирующих правил трансформации. Определяющие правила оказывают влияние лишь на последовательность применения модифицирующих правил, но сами по себе они не осуществляют преобразований в ходе трансформации.

При работе алгоритма трансформации описание всех шагов алгоритма сохраняется в так называемой «трейс-информации». Однако, трейс-информация во время работы алгоритма сохраняется в виде символьной строки, а после окончания работы алгоритма выводится на экран и удаляется из оперативной памяти, что делает ее неудобной для использования в алгоритме корректировки правил. Поэтому к алгоритму трансформации был добавлен алгоритм сохранения необходимой информации (в том числе информации обо всех промежуточных конструкциях, полученных в процессе трансформации).

Таким образом, весь процесс рассматриваемой трансформации представляется в виде последовательности конструкций $Q_1, Q_2, \dots, Q_m$, где Q_1- исходное предложение на языке UNL, Q_m- предложение на естественном языке, являющееся результатом трансформации; $Q_2, Q_3, \dots, Q_{m-1}$ - промежуточные конструкции. При этом переход от конструкции Q_k к

конструкции Q_{k+1} при $1 \le k < m$ осуществляется, как было указано выше, следующим образом: некоторый подграф конструкции Q_k (обозначаемый ниже посредством U_k) заменяется некоторым подграфом, обозначаемым ниже посредством V_k.

Рассмотрим несколько подробнее структуру ориентированных графов Q_k. Узлы и отношения, принадлежащие графу Q_k, обозначим посредством $d_1^{(k)}, d_2^{(k)}, \dots, d_{t_k}^{(k)}$, где t_k – количество узлов и отношений в графе Q_k. Подграфу U_k соответствует некоторое подмножество множесства узлов и отношений графа Q_k; обозначим узлы и отношения, принадлежащие этому подмножеству, посредством $a_1^{(k)}, a_2^{(k)}, \dots, a_{r_k}^{(k)}$, где r_k – количество узлов и отношений в графе U_k.

При преобразовании подграфа U_k в подграф V_k посредством рассматриваемого модифицирующего правила каждому узлу или отношению $a_i^{(k)}$ при $1 \le i \le r_k$ соответствует некоторое множество узлов и\или отношений $A_i^{(k)}$ подграфа V_k графа Q_{k+1} при $1 \le i \le r_k$. Отметим, что некоторые из множеств $A_i^{(k)}$ могут быть пустыми. Аналогично, объединение всех множеств $A_i^{(k)}$ при $1 \le i \le r_k$ может, вообще говоря, не совпадать с множеством узлов и отношений подграфа V_k. Те узлы и отношения, которые принадлежат подграфу V_k, но не принадлежат объединению множеств $A_i^{(k)}$ при $1 \le i \le r_k$, мы будем называть *новыми узлами и\или отношениями* при переходе от конструкции Q_k к конструкции Q_{k+1}.

Соответствие между узлами и отношениями $a_i^{(k)}$ и множеством $A_i^{(k)}$ после перехода к следующему шагу трансформации по умолчанию в системе не сохраняется, однако, как было отмечено выше, нами добавлен алгоритм для сохранения соответствующей информации.

Определение 3.1. Всякую пару $(a_i^{(k)}, b)$, где $b \in A_i^{(k)}$, будем называть условно *стрелкой перехода* (или просто *стрелкой*), от узла или отношения $a_i^{(k)}$ конструкции Q_k к узлу или отношению b конструкции Q_{k+1}.

Определение 3.2. Стрелку $(a_i^{(k)}, b)$, будем называть *статической*, если информация, приданная узлу или отношению $a_i^{(k)}$ совпадает с информацией, приданной узлу или отношению b (иначе говоря, в случае статической стрелки $(a_i^{(k)}, b)$ узел или отношение $a_i^{(k)}$ без изменений переходит в узел или отношение b).

Определение 3.3. Если при переходе от узла или отношения $a_i^{(k)}$ к узлу или отношению b происходит изменение информации приданной этим узлам и\или отношениям, то стрелку $(a_i^{(k)}, b)$ будем называть *динамической*.

Отметим, что в случае, когда узел или отношение $d_i^{(k)}$ конструкции Q_k не принадлежит подграфу U_k, то узел $d_i^{(k)}$ без изменений переходит в конструкцию Q_{k+1} в виде некоторого узла или отношения $d_j^{(k+1)}$. В этом случае пару $(d_i^{(k)}, d_j^{(k+1)})$, где $d_i^{(k)}$ и

$d_j^{(k+1)}$ равны друг другу, будем также называть стрелкой; будем считать ее *статической стрелкой*.

Определение 3.4. Если $(d_i^{(k)}, b)$ – стрелка, то узел или отношение b будем называть *непосредственным потомком* узла или отношения $d_i^{(k)}$; узел или отношение $d_i^{(k)}$ будем называть *непосредственным предком* узла или отношения b.

Определение 3.5. Пусть b – узел или отношение, принадлежащее конструкции Q_t, и пусть W – правило, применяемое при переходе от конструкции Q_k к конструкции Q_{k+1}, где $k < t$. Будем говорить, что правило W *повлияло* на узел или отношение b, если имеет место один из следующих двух случаев.

Случай 1: существует последовательность узлов и\или отношений $b^{(k)}, b^{(k+1)}, \dots, b^{(k+l)}$, такая, что каждый узел или отношение $b^{(k+i)}$ при $0 \le i \le l$ принадлежит конструкции Q_{k+i}, узел или отношение $b^{(k+l)}$ совпадает с b (таким образом, $k + l = t$), каждый узел вида $b^{(k+i+1)}$ является непосредственным потомком узла или отношения $b^{(k+i)}$, стрелка $(b^{(k)}, b^{(k+1)})$ является динамической, и все стрелки вида $(b^{(k+j)}, b^{(k+j+1)})$ при $j > 0$ являются статическими.

Случай 2: существует последовательность узлов и\или отношений $b^{(k)}, b^{(k+1)}, \dots, b^{(k+l)}$, такая, что каждый узел или отношение $b^{(k+i)}$ при $0 \le i \le l$ принадлежит конструкции Q_{k+i}, узел $b^{(k+l)}$ совпадает с b (таким образом, $k + l = t$), каждый узел или отношение вида $b^{(k+i+1)}$ является непосредственным потомком узла или отношения $b^{(k+i)}$, узел или отношение $b^{(k+1)}$ является новым узлом или отношением при переходе от Q_k к Q_{k+1}, и все стрелки вида $(b^{(k+j)}, b^{(k+j+1)})$ при $j > 0$ являются статическими.

Определение 3.6. Будем говорить, что правило W *косвенно повлияло* на узел или отношение b, если имеет место один из двух случаев, задаваемых такими же условиями, как и в предыдущем определении, с той лишь разницей, что условие «стрелки вида $(b^{(k+j)}, b^{(k+j+1)})$ являются статическими» заменяется проивоположным условием «хотя бы одна из стрелок $(b^{(k+j)}, b^{(k+j+1)})$ при $j > 0$ является динамической»

Для простоты изложения, шаг алгоритма трансформации назовем *повлиявшим* на данный узел либо отношение, если правило, примененное на данном шаге, повлияло на данный узел либо отношение.

Определение 3.7. Будем говорить, что правило W *ответственно* за возникновение подграфа G в некоторой конструкции Q_k, если W *повлияло* на узлы и\или отношения принадлежащие множеству F, где F некоторое непустое подмножество вершин или отношений подграфа G.

Определение 3.8. Будем говорить, что правило W *косвенно ответственно* за возникновение подграфа G в некоторой конструкции Q_k, если W *косвенно повлияло* на узлы и\или отношения принадлежащие множеству F, где F - некоторое непустое подмножество вершин или отношений подграфа G.

Отметим, что за некоторый данный подграф ответственность и косвенную ответственность обычно несут сразу несколько правил.

Исходя из этих определений поставленную задачу можно сформулировать как поиск ответственных и косвенно ответственных правил и их корректировка, либо создание новых ответственных и\или косвенно ответственных правил.

Далее, в этом разделе более детально рассматривается механизм применения правил, используемый в алгоритме трансформации. Будем использовать принятые выше обозначения, а именно: на k-ом шаге трансформации, при применении модифицирующего правила W_k, происходит переход от конструкции Q_k к конструкции Q_{k+1} при $1 \leq k < m$, где m - общее количество шагов трансформации. При этом, как было указано выше, некоторый подграф U_k конструкции Q_k заменяется некоторым подграфом V_k в конструкции Q_{k+1}.

В алгоритме трансформации применение правила происходит следующим образом. При применении вызывается функция *match*, которая проверяет конструкцию Q_k на наличие в ней подграфа U_k , удовлетворяющего условию правила. Эта функция возвращает объект класса *MatchSet*. Возвращаемый объект пустой, если нет подграфа U_k, соответствующего условию правила. В случае нахождение удовлетворяющему условию подграфа U_k, возращаемый объект содержит копию данного подграфа.

Если найден подграф U_k, удовлетворяющий условию правила W_k, происходит применение этого правила, при помощи вызова функции *execute*. Данная функция возвращает объект типа *OutputSet*, который содержит копию подграфа V_k, полученного применением правила W_k.

Фактически объект класса *MatchSet* содержит копию подграфа U_k, который удовлетворяют условию правила W_k, а объект класса *OutputSet* содержит копию подграфа V_k, который получаются при применении правила W_k.

Как было сказано выше, в ходе трансформации сохраняется необходимая информация о применении правил. Поэтому объекты *MatchSet* и *OutputSet* сохраняются на каждом шаге трансформации.

Предлагаемый подход к решению задачи реализуется при помощи двух составных частей: алгоритма нахождения ответственных правил и алгоритмов корректировки правил при соответствующих ошибках (модулей для корректировки правил).

Алгоритм нахождения ответственных правил в рамках данного решения также называется базовым алгоритмом. Базовый алгоритм обеспечивает получение необходимых данных для определения свойств ошибки и ее дальнейшей корректировки, и, следовательно, его применение обязательно для всех типов возможных ошибок.

Алгоритмы корректировки правил в рамках данного решения называются модулями. Модули являются независимыми друг от друга алгоритмами и имеют одинаковый программный интерфейс. Каждый модуль предназначен для корректировки определенного типа ошибок.

Базовый алгоритм обходит все шаги трансформации и среди них выделяет повлиявшие на данный узел шаги. Далее базовый алгоритм принимает решение, о том,

какой модуль должен быть вызван, подготавливает данные для передачи модулю и производит вызов модуля с передачей соответствующих данных. Далее модуль начинает обработку полученных данных.

С целью решения задачи сохранения необходимой информацию о шагах трансформации в ходе работы алгоритма трансформации, алгоритм нахождения ответственных правил (базовый алгоритм) разделяется на два этапа.

Первый этап работы алгоритма проводится еще в процессе работы алгоритма трансформации, благодаря чему необходимая информация сохраняется в соответствующем формате.

На втором этапе работы базового алгоритма, который имеет место при обнаружении ошибки пользователем и запуска процесса ее корректировки, анализируются данные, накопленные в ходе трансформации, и выделяются соответствующие шаги трансформации, содержащие ответственные правила.

Первый этап работы базового алгоритма реализуется при помощи программного класса *Tracer*, который позволяет хранить данные, необходимые для определения ответственного правила. Как было отмечено выше, существенное значение в процессе трансформации играют объекты типа *MatchSet* и *OutputSet*, генерируемые на каждом шаге алгоритма трансформации и содержащие копии подграфов U_k и V_k. На каждом шаге трансформации при применении соответствующего правила в объект типа *TreeNode* сохраняется *MatchSet*, *OutputSet*, применяемое правило W_k, а также конструкция Q_{k+1}, полученная в результате его применения. Класс *Tracer* содержит объект *Path*, который реализован хеш-таблицей, ключом которой является номер примененного правила (то есть k), а соответствующим значением - объект типа *TreeNode*. Соответственно, на каждом шаге трансформации в объект *Path* добавляется номер шага и созданный на этом шаге объект *TreeNode*. Тем самым после окончания процесса трансформации, в объекте *Tracer* будут содержаться данные обо всех шагах трансформации.

Второй этап базового алгоритма выполняется после завершения процесса трансформации, указания на ошибку и запуска процесса ее корректировки. На этом этапе происходит поиск шагов, повлиявших на данный узел. Как было сказано выше, после завершения процесса трансформации, в случае нахождения ошибки пользователь указывает узел, содержащий ошибку, и тип ошибки. Получая эти данные на вход вторая часть базового алгоритма начинает работу.

Вторая часть алгоритма нахождения ответственных правил состоит из внешнего цикла, который позволяет обходить все сохраненные ранее шаги трансформации в обратном порядке. На каждом шаге анализируются *OutputSet* и *MatchSet*, как это описано далее. Для нахождения ответственных правил выделяются собственные идентификаторы соответствующих узлов и передаются на вход функции *AnalyzePath*. Функция *AnalyzePath* при помощи цикла просматривает информацию содержащуюся в данном шаге *OutputSet* и *MatchSet*. Правило будет признано повлиявшим на узел, если данный узел содержиться в *OutputSet* и\или в *MatchSet*. Далее для каждого шага трансформации, содержащего повлиявшие на данный узел правила, создается объект класса *HistoryItem*,

который сохраняется в объекте класса *History*. На этой стадии выделяется подпоследовательность шагов, повлиявших на данный узел либо отношение. Это в свою очередь позволяет при использовании методов корректирования ошибок выделить шаги, ответственные за ошибку, среди всех повлиявших на данный узел либо отношение шагов.

Далее в главе дана оценка сложности алгоритма нахождения ответственных правил. Если принять максимальное на всех шагах трансформации совокупное количество узлов и отношений в предложении за M, а количество шагов при трансформации обозначить посредством N, тогда сложность алгоритма работы можно оценить числом O(2(N * 2M)).

Методы корректировки правил реализуются посредством нескольких алгоритмов. Каждый из этих алгоритмов предназначен для применения в случае возникновения определенного типа ошибок. Следует также учитывать, что алгоритмы корректировки правил могут предлагать пользователю внести корректировки не только в модифицирующие правила, но и в соответствующие записи в словаре. В главе описываются методы корректировки правил для описанных выше классов ошибок.

В четвертой главе описана программная среда для отладки наборов правил. Это понятие включает в себя: во-первых, метод хранения и обработки данных о применениях правил, имевших место в системе и приведших к корректным результатам; во-вторых, пользовательский интерфейс для работы с алгоритмами, описанными в работе.

В процессе работы по созданию и расширению набора правил трансформации, обычно проводимой специалистами в области лингвистики, часто возникают ситуации, когда правило, созданное или скорректированное для какого-либо случая, приводит к ошибке во ином случае. При корректировке подобных правила с целью избежать ошибки, получаемой во втором случае, важно также учитывать первоначальный случай.

С целью исключения возможных ошибок при повторных корректированиях правил и ускорению процесса отладки наборов правил нами было введено понятие *примеров применений правил*.

Примеры применений правил содержат в себе информацию об удачных применениях правила, то есть о тех применениях правила, при которых результат трансформации оценивался пользователем как корректный.

Разработан алгоритм, позволяющий редактировать существующее либо добавлять новое правило в тестовом режиме (без сохранения в базе данных) и протестировать его на записях, содержащих идентификатор данного правила, либо на всех записях в примерах применений правил. Этот алгоритм также называется *алгоритмом проверки совместимости правил*.

В главе также описан пользовательский интерфейс, встроенный в существующую систему UNDL Platform и обеспечивающий пользователю доступ к алгоритмам, описанным в работе.

В пятой главе приведены результаты экспериментов, проводимых с использованием UNL-корпуса для английского языка «TestDrive»[6] и набора правил для трансформации с языка UNL на армянский язык. В главе приведены результаты 23 тестов, в процессе которых была произведена корректировка как модифицирующих, так и определяющих правил. Приведены случаи повторного корректирования правил с учетом уже имевших место случаев редактирования данных правил. Тесты, при которых результаты трансформации были правильными, в главу включены не были.

В заключении диссертации резюмированы некоторые выводы из ее содержания, касающиеся результативности методов корректирования и расширения наборов правил, а именно, выводы, полученные на основе тестирования системы на реальных лингвистических ресурсах.

Как было сказано выше, представленные в работе методы позволяют пользователям проводить процесс отладки наборов правил. Таким образом система реализует подход «обучения с учителем». Одним из возможных путей усовершенствования системы может стать уменьшение роли пользователя в процессе отладки и постепенный переход к самообучающейся системе.

Основные результаты диссертации отражены в следующих публикациях:

1. Levon R. Hakobyan, "A Method of Constructing and Using the Tree of Possible Transformations from UNL to the Natural Language", Mathematical Problems of Computer Science Volume 41, IIAP NAS RA, Yerevan, 2014, pp. 131-134.

2. Levon R. Hakobyan, Aram A. Avetisyan, "An Automated System for Correction of Rules of Transformation from Universal Networking Language into Natural Language", Mathematical Problems of Computer Science Volume 41, IIAP NAS RA, Yerevan, 2014, pp. 114-121.

3. Levon Hakobyan, "On a Method of Improving the Quality of UNL Based Machine Translations", The 23rd Meeting of Computational Linguistics In the Netherlands, University of Twente, Enschede, 2013, p. 26.

[6] Test Drive Corpus, UNDL Foundation, [Online]. Available: http://www.unlweb.net/wiki/EUGENE#Test_drive

Թեմայի արդիականությունը

Քանի որ UNL-ի արդյունավետ օգտագործումը՝ որպես մեքենայացված թարգմանությունների միջնորդ լեզվի, կախված է օգտագործվող ռեսուրսներից (բառարան, կանոնների հավաքածու), ապա մասնագետներին, որոնք աշխատում են նման ռեսուրսների ստեղծման, ընդլայնման և փորձարկման ուղղությամբ, կարևոր է ունենալ համապատասխան ծրագրավորման գործիքներ, որոնք կնպաստեն սխալների հանգող ոչ ճիշտ կանոնների և բառարանային նոթերի (գրառումների) բացահայտմանը և ըստ հնարավարության նաև կնշեն սխալների վերացման ճանապարհը:

Աշխատանքում ներկայացված է ծրագրային համակարգ, որը հնարավորություն է տալիս լուծել նմանատիպ խնդիրներ, մասնավորապես, իրականացնել կանոնների հավաքածուի կարգավորումը մեքենայացված ռեժիմով:

Նման խնդիրների լուծման համար մշակված են ալգորիթմներ և ծրագրեր, որոնք հնարավորություն են տալիս օգտագործողին գտնել կանոններն ու բառարանային գրառումները, որոնք հանգեցրել են այս կամ այն սխալին, իսկ որոշ դեպքերում ստանալ նաև խորհուրդներ սխալների վերագման հնարավոր ճանապարհների վերաբերյալ:

Կանոնների ուղղման ընթացքում զգալի արգելքը է հանդիսանում փոխարկման այլընտրանքային ուղիների հավաքածուի արդյունավետ ստացումը և առավել կիրառելի ուղիների բացահայտումը: Օգտագործողը ստիպված էր ինքնուրույն ընտրել համապատասխան կանոնները տվյալ քայլին և հիշել տեղեկություններ ստացված տարբերակների մասին:

Այդ խնդիրը լուծվում է ալգորիթմերի և, այսպես կոչված, «փոխարկման հնարավոր ուղիների ծառի» կառուցման ծրագրերի օգտագործմամբ: Համապատասխան ալգորիթմը հնարավորություն է տալիս օգտագործողին ավտոմատացված ռեժիմով ստանալ բոլոր էական փոխակերպման ճանապարհները և դրանց արդյունքները: Ալգորիթմը նաև հնարավորություն է տալի համեմատել ստացված արդյունքները էտալոնի հետ, որը առաջարկված է օգտագործողի կողմից, մեքենայացված թարգմանության գնահատման որակի ալգորիթմի միջոցով:

Ատենախոսության նպատակը

Ատենախոսության նպատակն է մշակել այնպիսի ալգորիթմեր և ծրագրեր, որոնք թույլ են տալիս արդյունավետ ընդլայնել և ուղղել դիտարկվող փոխարկման կանոնների հավաքածուները և բառարանային գրառումները: Մշակված ալգորիթմերը և ծրագրերը թույլ են տալիս գտնել սխալ կանոնները ու բառարանային գրառումները և առաջարկում են դրանց շտկման հնարավոր ուղղիները:

Գիտական նորույթ

1. Մշակվել են նոր ալգորիթմեր և ծրագրեր, որոնք թույլ են տալիս ստանալ այսպես կոչված փոխարկման հնարավոր ուղիների ծառը, ինչը հնարավորություն է տալիս ստանալ փոխարկման տարբեր հնարավոր ուղիների հավաքածուներ:

2. Մշակվել է փոխարկման ալգորիթմի աշխատանքի ընթացքում գոյացող սխալների դասակարգումը, որը թույլ է տալիս նշել տվյալ սխալի հատկությունները և օգտագործել համապատասխան ուղղման ալգորիթմ:

3. Մշակվել են ալգորիթմեր և ծրագրեր փոխարկման ընթացքում գոյացող՝ այս կամ այն սխալների համար պատասխանատու կանոնների որոնման և ուղղման համար: Դրանք նաև թույլ են տալիս ուղղել համապատասխան կանոնները:

Հետազոտության արդյունքների կիրառման նշանակությունը

Փոխարկման արդյունքում գոյացող սխալների ուղղման մեքենայացված մեթոդի կիրառումը մասնագետների կողմից, թույլ է տալիս արագացնել կանոնների հավաքածուների մշակման ընթացքը: Դա կնպաստի UNL-ով թարգմանությունների որակի լավացմանը:

Պաշտպանությանը ներկայացվող հիմնական դրույթները

1. Մշակվել են փոխարկման հնարավոր ուղիների ծառի կառուցման ալգորիթմները և ծրագրերը, այդ ուղիների կիրառման մեթոդը համակցված տարբերակում և ենթածառերի կառուցման դեպքում, ինչպես նաև ծառի կառուցման ընթացքում ի հայտ եկող արդյունքների (տերևների) համեմատման մեթոդը [1,3]:

2. Մշակվել են փոխարկման արդյունքներում գոյացող հնարավոր սխալների ուղղման ալգորիթմերը և ծրագրերը: Դրանցից են տվյալ հանգույցի վրա ազդող կանոնների ալգորիթմերը և ծրագրերը և համապատասխան տեսակների սխալների ուղղման ալգորիթմերը և ծրագրերը [2]:

3. Մշակվել են փոխարկման կանոնների կիրառման օրինակների պահպանման մեթոդները և այդ տեղեկությունների օգտագործումը կանոնների հավաքածուների ուղղման հետագա ընթացքում [1]:

ABSTRACT

Levon Hakobyan

Automated System for the Expansion and Correction of the Rule Sets for Transformation from UNL into natural language

General description of the thesis

Since the effectiveness of using UNL as an intermediate language for automatic translation depends on the use of resources (dictionaries, a set of rules, etc.), the specialists working on the creation, expansion and testing of such resources need the appropriate software tools that allow the detection of invalid rules and dictionary entries, leading to errors and, if possible, specify the path to eliminate these errors.

The thesis presents a software system that allows to solve such problems, in particular, make debugging rule sets in an automated mode.

To solve such problems developed algorithms and programs that allow the user to find the rules and dictionary entries, which led to a particular error, and, in some cases, get advice on how to correct the error.

Significant obstacles in the process of debugging rules are the difficulty of obtaining a set of alternative ways of transformation and identify the most suitable among them. For this user had to manually select the rules for applying at corresponding step of transformation and remember information about the production of variants.

This problem is solved by means of algorithms and software to build the so-called "tree of possible transformation paths". Appropriate algorithm allows the user to automatically get all the essential ways of transformation and their results. The algorithm also makes it possible to compare the obtained results using the algorithms for the evaluation of the quality of machine translations.

The aim of the thesis

The aim of the thesis is to develop algorithms and programs to effectively expand and correct transformation rule sets and dictionary entries. The algorithms and programs offer the possibility of finding incorrect rules and dictionary entries and offer the user the possible ways of their correction.

Scientific novelty

1. New algorithms and applications for the construction of the tree of the possible transformation paths are developed, which let user to obtain different possible transformation paths.

2. A classification for the possible errors of the transformation result is given. This classification let to describe properties of the possible errors and use corresponding algorithms to correct them.

3. An algorithm for the searching of the rule which affected on the node is developed. Also the algorithms for the fixing of the errors of the corresponding types are developed. These algorithms find the rules and dictionary entries which are responsive for the error and suggest some correction ways.

Applicability of the results

Developed methods for the automated correction of the errors in the results of the transformation let the specialists to debug the transformation rule sets of UNL more effectively. This leads to the better quality of machine translations based on UNL.

The following statements are presented for defense

1.The algorithms and programs for the construction of the tree of possible transformations, methods of its applying on combined cases and for subtrees construction and methods for the comparing of the results (leaves) of the tree [1,3].

2. The algorithms and programs for the correction of the errors in a transformation result which consist of algorithms and programs for the affected rules searching and the algorithms and programs for the correction of the errors of corresponding types [2].

3. The methods for storing of the transformation rules applying examples and using them for the following debugging of the transformation rule sets [1].