(19) Евразийское патентное ведомство

(11) 037156 (13) B1

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ЕВРАЗИЙСКОМУ ПАТЕНТУ

(45) Дата публикации и выдачи патента
2021.02.12

(21) Номер заявки
201800581

(22) Дата подачи заявки
2018.09.24

(51) Int. Cl. ***G06F 17/27*** (2006.01)
G06F 17/24 (2006.01)

(54) СПОСОБ ПОИСКА В ТЕКСТЕ СОВПАДЕНИЙ С ШАБЛОНАМИ

(43) **2020.03.31**
(96) **2018/EA/0075 (BY) 2018.09.24**

(71)(73) Заявитель и патентовладелец:
ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ "НЕЗАБУДКА СОФТВЕР" (BY)

(72) Изобретатель:
Сурков Дмитрий Андреевич, Сурков Кирилл Андреевич, Четырько Юрий Михайлович, Шимко Иван Владимирович, Савёнок Владислав Александрович (BY)

(74) Представитель:
Сурков Д.А. (BY)

(56) US-A1-20130080886
WO-A3-2008103398
US-B2-7054855
US-A1-20110295595
US-B2-8155946
RU-C2-2605077

(57) Изобретение относится к способу поиска в тексте совпадений с шаблонами и обеспечивает быстрое выявление набора понятий, сущностей и отношений в текстах на естественном языке. Техническим результатом является повышение скорости, полноты и точности поиска, обеспечение безопасности поиска и независимости от языка. Текст, в котором выполняется поиск, разбирают на лексемы. На языке описания шаблонов создают набор шаблонов в виде последовательностей, вариаций и повторений лексем текста и вхождений других шаблонов. Транслируют набор шаблонов в поисковые выражения с поисковыми индексами. Поиск всех совпадений всех шаблонов выполняют за один последовательный просмотр лексем текста. По каждой лексеме в поисковых индексах отыскивают поисковые выражения, начинающиеся с этой лексемы, создают кандидатов совпадений для соответствующих шаблонов и заносят их в набор кандидатов. По мере просмотра лексем из набора кандидатов изымают совпадения и несовпадения.

037156 B1

037156 B1

Область техники, к которой относится изобретение

Изобретение относится к способу поиска в тексте совпадений с шаблонами и обеспечивает быстрое выявление набора понятий, сущностей и их отношений в текстах на естественном языке.

Предшествующий уровень техники

В настоящее время многие прикладные программные системы, предназначенные для автоматизации разных отраслей человеческой деятельности, включают в себя средства анализа текста на естественном языке. Источниками текстовых данных в таких системах являются базы данных, хранилища документов, публикации на сайтах в сети интернет, записи в блогах и на форумах, сообщения электронной почты, сообщения социальных сетей, стенограммы из центров обработки звонков, сообщения бесед в программах текстовой связи, и другие системы. Задача анализа текстовых данных в таких системах заключается в выявлении понятий и сущностей в тексте, а также в определении отношений между ними. Анализ текстовых данных часто сопровождается этапом обогащения текстовых данных метаинформацией о найденных понятиях, сущностях и отношениях, совокупно называемых тегами. Процесс обогащения текстовых данных тегами известен как тегирование текста.

Задача анализа текстовых данных в настоящее время решается различными способами, среди которых наиболее известны и применяются полнотекстовый поиск, полнотекстовая индексация, методы машинного обучения, латентный семантический анализ, анализ на основе регулярных выражений, анализ на основе формальных грамматик с учётом или без учёта морфологии слов, а также различные комбинации перечисленных способов.

Ранним подходом к решению задачи анализа текста является полнотекстовый поиск - автоматизированный документальный поиск заданных комбинаций ключевых слов и выражений в полном тексте или в существенной части текста документа. В процессе полнотекстового поиска обычно присутствуют следующие подготовительные этапы: разбор текста на слова, удаление шумовых слов, выделение основ слов, приведение слов к словарной форме, выполнение статистического анализа. При этом часто учитывают морфологию языка анализируемого текста. Для разбора текста на слова используют подходы разной степени сложности, среди которых известен стандарт Unicode Standard Annex #29 - определяет наиболее точные правила для разбора текста на предложения и слова.

Для существенного ускорения полнотекстового поиска в случае его применения к большим объёмам текстовых данных выполняют полнотекстовую индексацию, т.е. создают поисковый индекс. Поисковый индекс - это структура данных, называемая словарём, в которой хранятся все слова текста вместе с информацией о том, в каких местах каких документов они встречаются. Полнотекстовая индексация не является обязательным этапом полнотекстового поиска, который может выполняться путём нахождения подстроки в строке с помощью известных алгоритмов, однако позволяет значительно сократить область, в которой выполняется поиск, а значит существенно ускорить поиск по сравнению с прямым поиском в тексте исходных документов. При построении поискового индекса обычно игнорируют так называемые шумовые слова - слова, которые содержатся практически во всех документах К таким словам часто относят предлоги, причастия, междометия, частицы и др. Шумовые слова, как правило, исключают и из поисковых запросов.

Для уменьшения вариативности словоформ естественного языка применяют стемминг - процесс нахождения основы слова для заданного исходного слова. Стемминг позволяет устранить необходимость перечисления словоформ в запросах и правилах поиска, однако многозначность слов и несовершенство алгоритмов стемминга приводят к увеличению числа ложных совпадений.

Для выделения ключевых слов в процессе полнотекстовой индексации часто используют статистический анализ. Он предполагает подсчёт частоты использования слов в тексте и выделение наиболее часто используемых слов. Ключевым компонентом этого подхода является математическая модель, которую применяют при расчётах частоты использования слов.

Полнотекстовый поиск отличается высокой производительностью и хорошей масштабируемостью в распределённой вычислительной среде. Однако этот подход малоэффективен при решении сложных задач анализа текста из-за невысокой точности поиска и отсутствия возможности поиска понятий, сущностей и их отношений, так как ориентирован на поиск ключевых слов. Указанный недостаток не позволяет использовать полнотекстовый поиск для тегирования текста.

В настоящее время для анализа текста на естественном языке широко применяют методы машинного обучения. Эти методы основаны на классификации объектов, представленных описаниями в определённом пространстве признаков. Выделяют две группы методов машинного обучения: обучение с учителем и обучение без учителя. В первой группе целью обучения является получение правил, с помощью которых можно произвести классификацию новых объектов на основании тех, которые использовались для обучения. Во второй группе обучающая выборка отсутствует, и стоит задача кластеризации - объединения объектов в группы на основании заданной меры их сходства или различия. Машинное обучение активно применяют для выделения ключевых слов, выявления сущностей и отношений, а также распознавания эмоций, тематики и языка текста.

Среди методов машинного обучения известны те, что ориентированы на построение предметной онтологии с информацией об объектах и их отношениях. В качестве объектов выступают существитель-

ные, а в качестве отношений - глаголы, с которыми ассоциированы существительные. Известны методы машинного обучения, которые выявляют такие смысловые зависимости с помощью математических моделей нейронных сетей.

Среди методов анализа текста на естественном языке известен латентный семантический анализ, который решает задачу поиска сходства текстов через сходство значений слов в контексте их использования. Значения слов считаются сходными, если слова используются в сходных контекстах.

Применение методов машинного обучения требует следующих подготовительных этапов обработки текста: разбор текста на лексемы, нормализация слов и морфологический анализ. Далее следует фаза грамматического анализа, в ходе которого активно применяются методы машинного обучения. Грамматический анализ подразделяется на поверхностный и полный. Поверхностный анализ способен выявлять некоторые смысловые составляющие в тексте: именные, глагольные и предложные группы. Полный грамматический анализ представляет структуру всего предложения в виде грамматического дерева. В ходе полного грамматического разбора часто возникают неоднозначности, решение которых является достаточно трудоёмкой задачей. Кроме того, с практической точки зрения для получения приемлемого результата анализа необходимость в полном грамматическом разборе отсутствует.

Главным недостатком методов машинного обучения, который вытекает из сути данного подхода, является невозможность объяснения полученных результатов. Это вызвано природой математических моделей и процессом обучения классификатора, который происходит путём сложных модификаций числовых коэффициентов на основании обучающей выборки. Ещё одним недостатком является сложность тренировки системы для распознавания новых сущностей и понятий, во-первых, это требует достаточно большого объёма новых данных для обучения; во-вторых, в результате обучения для распознавания новых понятий может понизиться точность распознавания понятий, добавленных ранее. Наконец, существенным недостатком методов машинного обучения является низкая производительность. Как первичное обучение, так и классификация требуют много времени, что затрудняет использование данного подхода для быстрого тегирования текста, в частности для повторяющегося тегирования большого количества документов в базе данных.

В настоящее время для анализа текста широко применяются регулярные выражения, которые являются средством поиска шаблонов в тексте как на машинно-ориентированных, так и на естественных языках. Данный подход характеризуется хорошей объяснимостью результатов и относительной свободой в задании шаблонов поиска. Базовые понятия регулярных выражений включают в себя символьные литералы, а также их последовательности, чередования и множества. Большинство современных реализаций регулярных выражений дополнительно позволяют указывать квантификаторы, выполнять просмотр вперёд и назад, а также обеспечивают поиск по классам символов.

Основным недостатком регулярных выражений является отсутствие оптимизации для поиска множества шаблонов за один проход по тексту. При увеличении числа шаблонов производительность линейно снижается, так как поиск каждого шаблона требует отдельного просмотра текста. Ещё одним недостатком данного подхода является работа с текстовыми данными на уровне символов, а не слов или лексем, а также невозможность использования в шаблонах ссылок на другие шаблоны, в том числе рекурсивно определённых шаблонов, что снижает скорость, точность и полноту анализа текста на естественном языке. Ещё одним недостатком регулярных выражений является существование для отдельных шаблонов определённых наборов входных данных, на которых перебор всех вариантов совпадения приводит к частым возвратам назад по тексту и сильному замедлению поиска, что воспринимается пользователем как бесконечное зацикливание. Существенным недостатком регулярных выражений является также низкая читаемость описаний шаблонов - при записи выражения высока вероятность допущения ошибки, затруднено понимание написанных ранее шаблонов.

Более поздние подходы, применяемые для анализа текста на естественном языке, основаны на использовании наборов формальных грамматик. Правила определения формальных грамматик предполагают использование в левой части выражения одного нетерминального символа, а в правой части терминальных и нетерминальных символов, связанных некоторыми операторами. К терминальным символам (терминалам) относятся объекты, слова и символы языка, имеющие конкретное символьное значение. К нетерминальным символам (нетерминалам) относятся объекты, обозначающие какую-либо сущность языка и не имеющие конкретного символьного значения. Операторы связывают терминалы и нетерминалы в последовательности (цепочки), множества альтернатив (вариации), повторения и другие конструкции.

Известным примером формальных грамматик является форма Бэкуса-Наура. В данной формальной системе в правой части правила допускается указание последовательности и вариативности терминалов и нетерминалов. Также известен ряд модификаций формы Бэкуса-Наура, которые позволяют описывать повторения и необязательные элементы. В них существует возможность указать как возможность пропуска или повторения отдельной части выражения произвольное число раз, так и точное число повторений с указанием диапазона значений повторителя.

Хотя синтаксис грамматик в форме Бэкуса-Наура является более простым для восприятия по сравнению с синтаксисом регулярных выражений, грамматики этого вида ориентированы на разбор машин-

но-ориентированного текста в корневой нетереминал и плохо пригодны для задач анализа и тегирования текста на естественном языке.

Известным примером использования формальных грамматик для анализа текстов на естественном языке является язык шаблонов JAPE и система на его основе. Язык JAPE представляет из себя язык правил для выполнения текстовой разметки в форме аннотаций, приписываемых к непрерывным фрагментам обрабатываемого текста. Анализ текста предполагает выполнение следующих этапов:

разбор текста на лексемы: числа, знаки пунктуации, символы, знаки пробелов. Тип лексемы сохраняется в соответствующую аннотацию;

выделение именованных сущностей в соответствии с их списками в текстовом файле с помощью так называемого газетира. Каждый список содержит отдельное множество сущностей: организации, города, имена и так далее;

разделение текста на предложения. При выделении предложений используется список аббревиатур из газетира, чтобы отличать конец предложения от других случаев использования знаков препинания;

разделение текста на предложения с использованием регулярных выражений;

определение частей речи. Используется встроенный набор правил, выявленные части речи сохраняются в аннотации;

семантическая аннотация, использующая сформулированные правила на языке JAPE и аннотации, полученные в результате выполнения предыдущих этапов.

Грамматика на языке JAPE состоит из последовательности этапов, каждый из которых представляет собой набор правил, описывающих действия, выполняемые для некоторого шаблона текста. Наборы правил составляют каскад преобразователей аннотаций, которые запускаются последовательно. В левой части правила приводится описание шаблона, а в правой перечисляются операторы преобразования аннотации. Аннотации, совпавшие в шаблоне из левой части выражения, могут быть использованы в правой части путём ссылок на метки, которые ассоциированы с элементами шаблона. Левая часть правила представляет собой шаблон из аннотаций, в которых можно использовать свойства из других аннотаций, особые свойства, содержащие непосредственно аннотированную строку текста и её длину, а также операторы сравнения и регулярные выражения для строк. Шаблоны допустимо объединять в последовательности и альтернативы, а также задавать повторения элементов шаблона. Можно задавать несколько комбинаций шаблона и действия над аннотацией в одном правиле. Для этого шаблонам задают уникальные метки, используемые в дальнейшем в правой части правил. Помеченные метками шаблоны можно включать друг в друга и создавать иерархические аннотации. В левой части правила допустимо применение так называемых макросов, т.е. подстановочных элементов, которые можно многократно использовать при записи правил. В правой части правила допустимо использовать код на языке программирования Java, позволяющий реализовывать сложную логику операций над аннотациями. Такой код включается в сгенерированный на основании грамматики программный Java-класс.

Подход к анализу текста на основе языка JAPE является достаточно универсальным и расширяемым, однако платой за это является громоздкость получаемых шаблонов поиска и высокая сложность всей системы. Наиболее существенным недостатком является ориентация на многопроходную последовательную проверку правил, что подтверждается применением регулярных выражений и программного кода Java, и как следствие, линейное снижение производительности при увеличении числа шаблонов, а также небезопасность в смысле возможности возникновения слишком глубоких рекурсий и (условно) бесконечного зацикливания поиска.

Известен подход для анализа текста на естественном языке, учитывающий морфологию. Примером реализации этого подхода является язык лексико-синтаксических шаблонов LSPL и система на его основе. Язык LSPL является декларативным языком для спецификации лексических и грамматических свойств конструкций, выделяемых в текстах на русском языке. Шаблон в этом языке задаётся указанием его имени, за которым через знак равенства следует, в общем случае, описание нескольких альтернативных вариантов формализуемой языковой конструкции. Для разделения альтернатив используется символ "|". Разные шаблоны должны иметь разные имена, а в качестве имени должна использоваться произвольная последовательность букв, причём первая буква должна быть заглавной. Последовательность элементов в шаблоне строго соответствует их расположению в описываемой конструкции. Элементами шаблона могут выступать элемент-слово, элемент-строка, экземпляр шаблона, повторение элементов и набор альтернатив. Первые три вида элементов относятся к простым элементам, а остальные - к сложным, так как в их состав входят другие элементы. Необязательные элементы, повторения и альтернативы аналогичны операторам в языке JAPE и дополненной форме Бэкуса-Наура. Элемент-строка позволяет записать в шаблоне конкретную строку символов, например конкретную форму слова, знак пунктуации или условное обозначение. В таких строках можно использовать регулярные выражения. При этом в элементах-строках не допускается уточнение морфологических признаков слов. Элемент-слово соответствует отдельному слову, для которого указываются следующие признаки:

часть речи: при этом используются символьные обозначения: N - существительное, V - глагол, A - прилагательное, Pr - предлог, Pn - местоимение и т.д.;

имя лексемы - начальная форма слова, задающая множества всех словоформ этого слова;

морфологические характеристики слова: падеж, род, число (единственное или множественное) и др.

Каждой части речи соответствует свой набор морфологических характеристик, причём некоторые из них фиксированы, например род существительных, другие же изменяемы, например падеж существительного. Язык LSPL допускает использование в шаблонах произвольных слов, обозначаемых буквой W. Конкретные значения изменяемых характеристик могут быть записаны в угловых скобках после имени лексемы. Например, следующий шаблон задаёт прилагательное красный в именительном падеже женского рода:

```
A<(красный),c=nom,g=fem>
```

здесь A - прилагательное (от англ. adjective), далее в угловых скобках характеристики прилагательного, где "красный" - начальная форма слова, c=nom - обозначение именительного падежа (c от англ. case - падеж, n от англ. nominative - именительный), g=fem - обозначение женского рода (g от англ. gender - род, fern от англ. feminine - женский).

В шаблоне допускается задание повторения элементов: в фигурных скобках указываются элементы, которые могут встречаться в тексте несколько раз подряд. Для повторений возможно указание максимального и минимального множителя в угловых скобках после шаблона. Так, запись {A}<1,3> обозначает прилагательное, которое может повторяться от одного до трёх раз. Если второе значение не указано, зафиксированным считается только минимальное число повторений. Стоит отметить, что такой подход к использованию значения по умолчанию является не совсем очевидным и требует обязательных разъяснений для пользователя. Существует возможность задавать необязательные (опциональные) элементы, помещая их в квадратные скобки. Такая запись является укороченной версией повторения от нуля до одного раза: например, элемент ["не"] указывает необязательность совпадения строки "не" с участком текста. Опциональные элементы и повторения могут состоять из набора альтернатив, каждая из которых, в свою очередь, может быть последовательностью.

Условия согласования определяют грамматическое согласование отдельных элементов шаблона и записываются в угловых скобках после всех согласуемых элементов. Условие согласования выражается в виде равенства отдельных или всех общих морфологических характеристик согласуемых слов. Например, следующий шаблон описывает согласованную в числе и роде пару слов - местоимение и глагол:

```
PV = Pn V <Pn.n=V.n, Pn.g=V.g>
```

Здесь Pn - местоимение (от англ. pronoun), V - глагол (от англ. verb), далее в угловых скобках следует описание условий согласования, где Pn.n=V.n (n от англ. number - число) задаёт согласование числа предлога и глагола, а Pn.g=V.g - рода.

В такой записи используются составные имена, образованные из имени элемента шаблона и имени согласуемого признака, которые разделяются точкой. В случае, когда должны быть согласованы все общие морфологические характеристики двух элементов шаблона, условия согласования можно записать, используя только имена согласуемых элементов, не перечисляя морфологические характеристики.

В случае, когда в шаблоне несколько элементов и несколько условий согласования, их можно записывать по очереди друг за другом. Единственным ограничением является то, что каждое условие согласования в шаблоне не должно опережать запись самих согласуемых элементов.

При описании шаблона часть морфологических характеристик слов можно вынести в параметры шаблона, которые записываются после всех его элементов и условий. В параметрах можно указать только морфологические характеристики входящих в шаблон элементов-слов, которые не были конкретизированы в самом шаблоне. Следующий пример показывает определение с помощью параметров шаблона падежа и числа существительного:

```
ANp = [A] N1 N2<c=nom> <A=N1> (N1.c, N1.n)
```

Здесь A - опциональное прилагательное, N1 и N2 - существительные, где для существительного N2 в угловых скобках задано ограничение c=nom - обозначение именительного падежа; далее в угловых скобках A=N1 задаёт грамматическое согласование всех общих морфологических характеристик прилагательного A и существительного N1, после чего в круглых скобках описаны морфологические характеристики первого существительного, выделяемые в качестве параметров шаблона ANp: c - падеж и n - число. При этом в качестве параметров шаблона нельзя использовать падеж существительного N2, так как он уже конкретизирован в самом шаблоне.

Для объявленного таким образом шаблона с параметрами можно в дальнейшем создавать экземпляры, фиксируя требуемые аргументы, для использования в других шаблонах. Например, для создания экземпляра приведённого шаблона ANp с заданным именительным падежом и единственным числом для первого существительного следует использовать конструкцию

```
ANp<c=nom, n=sing>
```

Здесь ANp - имя шаблона, для которого создаётся экземпляр, далее в угловых скобках c=nom - фиксация в качестве первого параметра шаблона именительного падежа, n=sing - фиксация в качестве второго параметра шаблона единственного числа (от англ. singular).

Для проверки наличия слов текста в некотором словаре в шаблонах допускается запись словарных условий, имеющих вид обращения к некоторой логической функции, проверяющей вхождение в словарь.

Имя функции в таких выражениях можно считать именем словаря, в котором выполняется поиск. Как и условия согласования, словарные условия записываются в угловых скобках. Например, запись <Dict(A1)> означает, что элемент-слово A1 должно входить в словарь Dict.

Набор взаимосвязанных шаблонов задаёт расширенную дополнительными условиями формальную грамматику. Для распознавания шаблонов последовательно применяется процедура наложения шаблона на текст, результатом которой являются различные варианты наложения. Вариант наложения - это найденный непрерывный фрагмент текста, соответствующий шаблону вместе с набором конкретных значений морфологических характеристик слов, входящих в этот отрезок. Значениями элементов шаблона являются указанные отрезки текста. В некоторых случаях возможны совпадения одного и того же шаблона с одним и тем же участком текста разными способами, а также пересечения совпадений.

Язык лексико-синтаксических шаблонов LSPL содержит богатые средства для учёта морфологических характеристик слов и их грамматического согласования. В отличие от языка JAPE, синтаксис LSPL менее громоздкий и относительно более понятный, но выраженная лингвистическая направленность данного подхода отрицательно влияет на простоту использования и производительность анализа. Ориентированность на русский язык является не столько недостатком, сколько вынужденной мерой, так как учёт морфологии требует тесной связи с конкретным языком. Выявление шаблонов происходит последовательно, что приводит к линейному снижению производительности при увеличении числа шаблонов для поиска. Использование регулярных выражений хотя и помогает увеличить точность и полноту анализа, но снижает безопасность подхода в смысле исключения слишком долгих переборов вариантов совпадений.

Глубокий анализ морфологических особенностей языка является важным направлением научных исследований, однако слабо приближает к решению задачи тегирования текста: грамматическое согласование слов оказывает влияние на незначительный процент результатов, так как анализируемые реальные тексты на естественном языке не составляются с целью затруднения работы алгоритмов поиска. Отбрасывание находящихся рядом и подходящих под шаблон по своим основам, но грамматически несогласованных слов, ощутимо не увеличивает качество поиска. Таким образом, для упрощения составления шаблонов для языков с богатым словообразованием достаточно использовать стемминг, а в некоторых случаях и просто перечислить требуемые формы слов.

Ещё одним известным способом анализа текста является подход на основе грамматик Томита-парсера. Этот подход схож с подходом на основе языка LSPL и использует словари ключевых слов и формальные грамматики, учитывающие морфологические признаки слов, для извлечения фактов из текста на естественном языке. Правила грамматик Томита-парсера работают с цепочками. Одна цепочка соответствует одному предложению в тексте. Из цепочки выделяются подцепочки, которые в свою очередь интерпретируются как факты, разделённые по полям. Например, правило

```
S -> Noun;
```

описывает существительное. При этом по умолчанию происходит приведение слов к начальной форме. В приведённом правиле Noun является терминалом. В Томита-парсере к терминалам относятся названия частей речи, символы пунктуации и арифметические знаки, слова в начальной форме - леммы, а также некоторые другие специальные терминалы, например терминал Word, обозначающий любое слово. Терминалы могут стоять только в правой части правил. В левой части правил записываются нетерминалы. В приведенном примере S - нетерминал. Если нетерминал встречается только в левой части правил, то он считается вершиной грамматики - начальным нетерминалом. При этом начальный нетерминал можно задать и явно. Для использования в правой части правил последовательности терминалов и нетерминалов они разделяются пробелами. Например, следующим образом определяется правило, выделяющее прилагательное и идущее следом за ним существительное:

```
S -> Adj Noun;
```

Для наложения ограничений на терминал или нетерминал в Томита-парсере используются специальные атрибуты, задаваемые в угловых скобках после соответствующего терминала или нетерминала. Например, для указания того, что некоторое слово должно начинаться с заглавной буквы, применяется атрибут h-reg1. В дополнение к этому атрибуту существуют атрибуты, обозначающие, что первая буква слова и как минимум ещё одна буква слова должны быть заглавными, или что слово должно быть полностью записано прописными буквами. Для этого используются атрибуты h-reg2 и l-reg соответственно.

Оператор повторения в языке описания грамматик Томита-парсера обозначается знаком "+" и означает, что терминал или нетерминал может встретиться в цепочке один или более раз. Для обозначения необязательности элемент заключается в круглые скобки. Например, следование за прилагательным нескольких слов с большой буквы, перед которыми может быть предлог, описывается шаблоном

```
S -> Adj (Prep) Word<h-reg1>+;
```

Здесь Adj - прилагательное, Prep - предлог, Word - любое слово, далее в угловых скобках h-reg1 задаёт, что слово Word должно начинаться с заглавной буквы, а символ "+" определяет, что слово Word может повторяться один и более раз.

В языке описания грамматик Томита-парсера возможно использование оператора альтернативы,

обозначаемого символом "|". Например, варианты обращения к человеку описываются шаблоном

```
FormOfAddress -> 'товарищ' | 'мистер' | 'господин';
```

Томита-парсер, как и язык лексико-синтаксических шаблонов LSPL, обладает средствами анализа морфологии и указания грамматического согласования слов. Грамматическое согласование задаётся специальными атрибутами, например, для согласования по роду, числу и падежу служит атрибут gnc-agr (от англ. gender, number, case agreement).

В каждой цепочке, определяемой правилом Томита-парсер, есть главное слово, грамматические признаки которого наследуются всей цепочкой. По умолчанию главным словом устанавливается первое слово цепочки, однако его можно указать вручную с помощью атрибута rt. Это может потребоваться из-за того, что нормализация слов выполняется исходя из заданного главного слова. В случае нормализации прилагательного и связанного с ним существительного, если не указать главным словом существительное, то в результате нормализации прилагательное будет приведено к мужскому роду, что может не совпадать с родом существительного.

Томита-парсер позволяет не только задавать ограничения на грамматическое согласование, но и отдельно указывать требуемые морфологические характеристики слов. Морфологические характеристики задаются с помощью атрибутов, например, для поиска слова женского рода, ему следует задать атрибут gram со значением "жен". В данном атрибуте через запятую можно указывать и другие морфологические характеристики слов, такие как число, падеж, вид глагола, форма прилагательных и так далее.

Для работы на уровне символов с применением регулярных выражений Томита-парсер содержит соответствующие атрибуты wfrn, wff и wfl. Синтаксис регулярных выражений, применяемых в Томита-парсере, основан на синтаксисе таковых в языке Perl. Например, для определения даты можно использовать шаблон

```
Date -> AnyWord<wff=/[1-2]?[0-9]{1,3}\.?/>
```

Для некоторых грамматик требуется не только морфологическая, но и семантическая информация о словах из текста. Такую информацию можно задавать с помощью словарей. Словарь состоит из статей, каждая из которых имеет название и набор слов, входящих в статью. Слова можно перечислять как в файле описания словаря, так и в отдельном текстовом файле, указав его имя в файле описания словаря. После описания словаря для использования его необходимо импортировать в корневой словарь. После этого в атрибутах терминалов можно использовать атрибут kwtype, значение которого определяет имя словаря, в который должно входить соответствующее слово.

Томита-парсер выделяет цепочки текста на основе грамматик, чтобы затем интерпретировать цепочки как факты и получить структурированные данные. Типы фактов описываются в специальном формате и содержат название факта и описание имён и типов составляющих его полей. Для каждого поля указывается, обязательно ли оно должно быть заполнено, для этого используются ключевые слова optional и required. Описание того, как следует интерпретировать выделенную цепочку, содержится в грамматике. Для этого используется ключевое слово interp, после которого в скобках записывается имя факта и имя поля факта, в которое должна попасть соответствующая цепочка.

Выявление цепочек текста, подходящих под грамматики, в Томита-парсере выполняется с помощью GLR-парсера. GLR-парсер - это модифицированная версия алгоритма LR-парсера, предназначенная для разбора по недетерминированным и неоднозначным грамматикам. GLR-парсер, как и LR-парсер, основан на использовании специальной структуры данных - таблицы анализа, которая содержит синтаксис анализируемого языка. При этом таблица анализа LR-парсера допускает только один переход состояния, определяемый состоянием парсера и входным терминалом. Для обеспечения работы с недетерминированными и неоднозначными грамматиками таблица GLR-парсера может содержать несколько переходов. Возникновение конфликта решается разветвлением стека парсера на два или больше параллельных стека, последние элементы которых соответствуют каждому из возможных переходов. В дальнейшем входные символы используются для определения последующих переходов всех полученных ветвей. Если же для какой-то из ветвей на некотором терминале не найдётся перехода, то она будет удалена как ошибочная. В лучшем случае для полностью детерминированной грамматики GLR-парсер работает за линейное время, в худшем случае - за кубическое. Приведённое описание справедливо для одной грамматики, так как процесс проверки совпадения сводится к сворачиванию исходной строки в стартовый нетерминал грамматики. Из всех возможных вариантов совпадения грамматики выбираются те, которые покрывают больший участок текста. После построения синтаксического дерева Томита-парсер выполняет создание фактов в соответствии с заданными описаниями.

Томита-парсер обладает схожим функционалом с языком лексико-синтаксических шаблонов LSPL, позволяя задавать правила грамматического согласования слов и требуемые морфологические характеристики. Повышенное внимание к морфологии приводит в данном случае к аналогичному усложнению синтаксиса записи шаблонов. Томита-парсер хорошо справляется с задачей выявления фактов в тексте, однако в задаче тегирования текста поиск, как правило, осуществляется по неким конкретным интересующим словам, а не по синтаксическим конструкциям, таким как, например, глагольные или именные группы. Применение Томита-парсера для задачи тегирования затруднено громоздкостью и сложностью

синтаксиса шаблонов, а также невысокой производительностью, что обусловлено богатыми возможностями по работе с морфологией.

Подводя итог, можно констатировать, что известные подходы отвечают многим, но не всем требованиям, предъявляемым к средствам выявления набора понятий, сущностей и их отношений в текстах на естественном языке. В действительности требуется сбалансированное решение на основе шаблонов, которое обладает высокой скоростью за счёт поиска всего набора шаблонов за один проход по лексемам текста, не снижает производительность линейно при увеличении числа шаблонов для поиска, одновременно с этим предоставляет мощный синтаксис описания шаблонов в виде формальных грамматик, включающих средства определения последовательностей (цепочек), вариаций (альтернатив) и повторений лексем текста и вхождений других шаблонов, в том числе рекурсивных шаблонов, обеспечивает приемлемые точность и полноту поиска, независимость от языка, а также безопасность в смысле исключения слишком долгих переборов вариантов совпадений.

Сущность изобретения

Техническим результатом изобретения является повышение скорости поиска в тексте совпадений с шаблонами за счёт предварительного разбора текста на лексемы и проверки всех совпадений всех шаблонов за один последовательный просмотр лексем текста.

Ещё одним техническим результатом изобретения является повышение полноты и точности поиска в тексте по шаблонам за счёт использования в качестве шаблонов формальных грамматик, допускающих последовательности, вариации и повторения лексем текста и вхождений других шаблонов, в том числе грамматик неограниченного уровня вложенности, рекурсивных грамматик и параметризованных грамматик.

Ещё одним техническим результатом изобретения является обеспечение независимости поиска от языка текста за счёт отказа от морфологического анализа текста и использования в шаблонах вариаций для учёта словоформ.

Ещё одним техническим результатом изобретения является обеспечение безопасности поиска, что означает избежание затягивания поиска на продолжительное время, воспринимаемое пользователем как зацикливание или бесконечный поиск, за счёт ограничения количества создаваемых в процессе поиска кандидатов совпадений или за счёт ограничения объёма потребляемых кандидатами ресурсов компьютера.

Ещё одним техническим результатом изобретения является повышение скорости, полноты и точности выявления набора понятий, сущностей и их отношений в естественно-языковых текстах за ограниченное время и независимо от языка.

В соответствии с одним из аспектов настоящего изобретения перечисленные технические результаты достигаются за счёт выполнения следующих шагов: текст предварительно разбирают на лексемы, к которым относятся, по крайней мере, слова и разделители слов; на языке описания шаблонов создают набор шаблонов, в котором каждый шаблон является формальной грамматикой, состоящей, по крайней мере, из последовательностей и(или) вариаций, и(или) повторений лексем текста и(или) вхождений других шаблонов и допускающей параметры, предназначенные для обобщения шаблонов и(или) для уточнения результатов поиска; транслируют набор шаблонов в деревья поисковых выражений с поисковыми индексами, которые позволяют по заданной лексеме или заданному имени шаблона быстро отыскать все поисковые выражения, которые прямо или косвенно начинаются с заданной лексемы или вхождения шаблона; создают переменный набор кандидатов, каждый из которых хранит информацию о совпадении фрагмента текста с соответствующим поисковым выражением и служит для принятия решения о дальнейшем совпадении или несовпадении фрагмента текста с шаблоном; затем последовательно просматривают лексемы текста и для каждой лексемы выполняют, по крайней мере, следующие действия: в поисковых индексах отыскивают все шаблоны, начинающиеся с текущей лексемы, создают кандидатов для проверки совпадений текста с этими шаблонами и добавляют их в набор кандидатов; для каждого кандидата из набора кандидатов принимают решение о полном совпадении, несовпадении или неполном совпадении текста с соответствующим шаблоном в зависимости от значения текущей лексемы; для каждого совпадения шаблона принимают решение о полном совпадении, несовпадении или неполном совпадении кандидатов, имеющих ссылки на этот шаблон; для учёта различных вариантов совпадения текста с проверяемыми шаблонами создают и добавляют в набор кандидатов логические копии тех кандидатов, для которых возможны разные варианты совпадения с текстом до текущей лексемы включительно, причём логические копии кандидатов наследуют общее состояние, которое соответствует уже просмотренным лексемам текста, и отличаются в той части состояния, которая соответствует текущей лексеме текста; в том случае, если количество создаваемых в процессе поиска кандидатов и(или) объём потребляемых кандидатами ресурсов компьютера превышает некоторый заданный предел, то, во избежание затягивания поиска на продолжительное время, набор кандидатов сокращается удалением части или всех кандидатов, что эквивалентно обрыву текста и началу нового поиска с текущей лексемы.

Перечень чертежей

На фиг. 1 показана архитектура устройства;

на фиг. 2 - пример шаблона и соответствующее ему дерево выражений;

на фиг. 3 - блок-схема, поясняющая общий алгоритм работы исполнителя поиска;

на фиг. 4 - блок-схема, поясняющая общий алгоритм принятия решений по дереву кандидатов в зависимости от значения текущей лексемы;

на фиг. 5 - пошаговое изменение состояния исполнителя поиска на примере поиска шаблона с альтернативами и исключениями вариации в некоторой входной последовательности лексем.

Сведения, подтверждающие возможность осуществления изобретения

Сведения, подтверждающие возможность осуществления изобретения, изложены далее в разделах "Архитектура устройства", "Язык описания шаблонов", "Правила обработки текста", "Разбор входного текста на лексемы", "Представление шаблонов на уровне исполнителя поиска", "Трансляция текстового описания шаблонов", "Структуры данных кандидатов и результатов совпадений", "Алгоритм работы исполнителя поиска", "Общий алгоритм принятия решений", "Правила принятия решений", "Рекомендации по тестированию", "Испытания производительности", "Заключение".

Архитектура устройства.

Частью данного изобретения является изображённый на фиг. 1 вариант архитектуры устройства, выполненный авторами для осуществления изобретения. Согласно фиг. 1 типичное устройство включает в себя транслятор с языка описания шаблонов 12, лексический анализатор текста 19 и исполнитель поиска 21.

Шаблоны 11, написанные на языке описания шаблонов, загружаются из текстового файла и подаются на вход транслятора с языка описания шаблонов 12, который выполняет синтаксический разбор шаблонов с помощью синтаксического анализатора 13, а затем создаёт пакет шаблонов 15 с помощью генератора пакета шаблонов 14. В процессе своей работы транслятор 12 обращается к лексическому анализатору текста 19 для преобразования текстовых литералов, состоящих более чем из одной лексемы, в последовательность лексем.

Пакет шаблонов 15 представляет собой транслированные описания шаблонов в виде структур данных в оперативной памяти и включает в себя поисковые выражения с поисковыми индексами 16 и корневой поисковый индекс 17.

Перед выполнением поиска лексический анализатор текста 19 выполняет разбор входного текста 18 на лексемы. Полученная в результате разбора последовательность лексем 20 передаётся вместе с пакетом шаблонов 15 исполнителю поиска 21.

Исполнитель поиска 21 отыскивает в тексте все совпадения с шаблонами за один последовательный просмотр лексем текста 20. Для реализации данной возможности исполнитель поиска 21 создаёт и обновляет структуру данных, которая содержит набор кандидатов совпадений 22. По каждой лексеме исполнитель поиска 21 обновляет набор кандидатов 22 на основе правил принятия решений 23 о полном совпадении, несовпадении или неполном совпадении соответствующих шаблонов. Кандидаты полностью совпадающих с текстом шаблонов превращаются в результаты совпадений 24. Кандидаты не совпадающих с текстом шаблонов удаляются. Кандидаты частично совпадающих с текстом шаблонов остаются в наборе кандидатов 22.

Выполненный авторами вариант осуществления изобретения использует методологию объектно-ориентированного программирования (ООП), что сделано с целью лучшего структурирования данных и алгоритмов, а также с целью упрощения и ускорения реализации. Однако выбор методологии и языка программирования с поддержкой ООП не является обязательным для осуществления изобретения. С тем же результатом может быть использована методология структурного программирования или другая методология, а также любой язык императивного программирования, в том числе машинный язык (ассемблер).

Язык описания шаблонов.

Частью данного изобретения является один из вариантов языка описания шаблонов, который позволяет декларативно описывать искомые конструкции в тексте с помощью формальных грамматик. Грамматики перечисляются в тексте друг за другом. Каждая грамматика начинается с имени (идентификатора), затем записывается знак равенства, затем следует выражение, состоящее из текстовых литералов, типов лексем, ссылок на другие грамматики и операторов. Грамматика завершается символом точки с запятой. В общем случае выражение может содержать следующие элементы:

текстовые литералы, сравниваемые с учётом или без учёта регистра символов (т.е. без учёта заглавных и прописных букв);

типы лексем: буквенная, цифровая или буквенно-цифровая последовательность символов, знак препинания, небуквенный знак, пробельный символ или признак перевода строки;

последовательности со строгим следованием элементов;

альтернативы (вариации), представляющие собой перечисления элементов, допустимых в данной точке шаблона, а также элементов, совпадение которых недопустимо;

повторения любого элемента шаблона минимально и максимально допустимое число раз;

область действия шаблона, задающая необходимость совпадения одного шаблона внутри другого;

следование элементов через слова с указанием максимального и минимального допустимого числа слов между элементами;

ссылки на другие шаблоны, т.е. вхождения других шаблонов.

Допустимо произвольное сочетание вышеперечисленных элементов. Их синтаксис и примеры на языке описания шаблонов рассмотрены ниже.

Текстовые литералы.

Текстовый литерал представляет собой заключённую в одинарные или двойные кавычки последовательность символов в кодировке Unicode

"Minsk"

По умолчанию текст сравнивается без учёта регистра символов. Если необходимо задать текст, который сравнивается с учётом регистра символов, то после кавычки, завершающей текст, записывается восклицательный знак. Например

"Minsk"!

При трансляции шаблона каждый текстовый литерал разбирается на лексемы по такому же алгоритму, что и текст, в котором выполняется поиск. В результате, исходный текстовый литерал преобразуется в последовательность лексем.

Типы лексем.

Типы лексем применяются для поиска лексем без указания их точного строкового представления. Доступны следующие типы лексем:

Alpha - буквенная лексема (от англ. alphabetic), непрерывная последовательность буквенных символов;

Num - цифровая лексема (от англ. numeric), непрерывная последовательность цифр;

AlphaNum - буквенно-цифровая лексема, непрерывная последовательность букв и цифр, начинающаяся с буквы (от англ. alphanumeric);

NumAlpha - буквенно-цифровая лексема, непрерывная последовательность букв и цифр, начинающаяся с цифры (от англ. numeric-alpha);

Punct - знак препинания (от англ. punctuation), относятся все знаки, выполняющие функции разделения или выделения смысловых частей текста, предложений, словосочетаний, слов, частей слова, указания отношений между словами, указания на тип предложения, его эмоциональную окраску, а также некоторые иные функции;

Symbol - небуквенный знак, например: &, %, $, #, _, {,} и др.;

Space - пробел, табуляция или другой разделитель без начертания;

NewLine - признак новой строки;

Start - признак начала текста;

End - признак конца текста.

Использование типа лексемы осуществляется по идентификатору (имени). Все идентификаторы в языке описания шаблонов чувствительны к регистру символов.

Последовательности.

Строгое следование элементов задаётся оператором последовательности, где каждый элемент является шаблоном. Для объединения элементов в последовательность используется знак "+". Например, доменное имя .com можно задать шаблоном

```
"." + "com"
```

Приведённая запись эквивалентна текстовому литералу ".com". Однако последовательность, полученная разбиением текстового литерала на части без соблюдения правил разбора на лексемы, не эквивалентна исходному текстовому литералу. Например, замена строковой константы "Minsk", состоящей из одной лексемы, на последовательность букв

```
"M" + "i" + "n" + "s" + "k"
```

приведёт к тому, что совпадение с текстом "Minsk" выявлено не будет.

Вариации.

Для перечисления нескольких элементов, совпадение которых допустимо в данной точке шаблона, используется оператор вариации. Как и в случае с последовательностью, каждый элемент вариации является шаблоном. Оператор вариации также обладает возможностью задавать шаблоны, совпадение которых недопустимо. Такие элементы вариации называются исключениями. Исключения способны отменять совпадения альтернатив, которые начали совпадать с той же позиции в тексте, и при этом могут покрывать больший участок текста, чем сама альтернатива. Вариация задаётся перечислением в фигурных скобках через запятую всех альтернатив и исключений. При этом исключения помечаются префиксным знаком "~". Например, упрощённый вариант шаблона поиска точки как разделителя предложений можно записать следующим образом:

```
{".", ~".net", ~".com"}
```

Приведённый шаблон выявит точку в тексте "Sentence.", но не выявит её в тексте "Sentence.net".

Повторения.

Язык описания шаблонов позволяет задавать количество повторений любого самостоятельного элемента шаблона с помощью оператора повторений. Под самостоятельным элементом понимается та-

кой элемент, который сам по себе является шаблоном, т.е., например, элемент исключения в вариации (знак "~" и следующий за ним шаблон) не является самостоятельным элементом, но сам шаблон, следующий за знаком "~", является таковым. Повторение задаётся с помощью квадратных скобок, записанных перед повторяемым элементом. Внутри квадратных скобок указывается минимальное и максимальное допустимое число повторений, разделённых знаком "-". Например, повторение элемента X от трёх до пяти раз задаётся шаблоном

```
[3-5] X
```

Если минимальное и максимальное допустимое число повторений совпадают, то допустимо указывать одно число. Например, повторение элемента X ровно три раза задаётся шаблоном

```
[3] X
```

Если минимальное и максимальное допустимое число повторений не заданы явно, то подразумевается, что они равны единице. Максимальное ограничение может быть не задано, что разрешает сколь угодно большое число повторений. В этом случае за минимальным ограничением ставится знак "+". Например, повторение элемента X один или более раз задаётся шаблоном

```
[1+] X
```

Отсутствие максимального ограничения в записи трактуется исполнителем поиска как достаточно большое число, которое при этом не должно быть слишком велико, чтобы не приводить к сильному замедлению проверки шаблонов. В случае нескольких пересекающихся вариантов совпадения выбирается наибольшая последовательность, совпадение которой в тексте началось раньше.

Минимальное число может быть нулевым, тогда считается, что совпадение элемента необязательно, такие элементы называются необязательными (опциональными). В случае, если элемент не встречается вовсе или встречается один раз, можно использовать вопросительный знак - сокращённую форму записи повторения от нуля до одного раза

```
? X
```

Необязательные элементы оказывают влияние и на шаблон, в состав которого они входят. Например, последовательность необязательных элементов сама считается необязательной, также необязательной считается вариация, если хотя бы один из её элементов-альтернатив является необязательным.

Стандартные шаблоны.

В языке описания шаблонов существуют предопределённые или стандартные шаблоны, которые автоматически доступны для использования в любых пользовательских шаблонах. Стандартные шаблоны служат для короткой записи часто используемых конструкций. Существуют следующие стандартные шаблоны:

Any - любая лексема, исключая признаки начала и конца текста, эквивалентно вариации {Alpha, Num, AlphaNum, NumAlpha, Space, Punct, Symbol, NewLine};

Word - слово, эквивалентно вариации {Alpha, Num, AlphaNum, NumAlpha};

Blanks - любое количество пробелов или признаков конца строки, эквивалентно шаблону [1+] {Space, NewLine};

WordBreaks - любое количество разделителей слов, таких как пробелы, знаки пунктуации, небуквенные знаки и признаки конца строки, эквивалентно шаблону

```
[1+] {Space, Punct, Symbol, NewLine}.
```

Язык описания шаблонов не специфицирует реализацию стандартных шаблонов на уровне исполнителя поиска. Хотя их всех можно выразить через уже описанные операторы, из соображений повышения производительности допустима также отдельная реализация.

Область действия.

Для поиска совпадения одного шаблона внутри другого используется оператор области действия. Данный оператор позволяет выполнять поиск конструкций в пределах определённых частей текста, например, предложений или абзацев. Оператор области действия обозначается символом "@" и включает в себя левую и правую части. В левой части записывается шаблон, который необходимо найти, а в правой - шаблон, внутри которого должно быть найдено совпадение шаблона из левой части. Один шаблон считается совпавшим внутри другого, если левая граница совпадения первого шаблона находится не раньше в тексте, чем левая граница совпадения второго шаблона, а правая граница - не позже в тексте, чем правая граница совпадения второго шаблона. Например, для поиска совпадения шаблона X внутри шаблона Y следует использовать шаблон

```
X @ Y
```

Здесь вместо Y допускается использовать как имя другого объявленного шаблона, так и непосредственно сам шаблон, объявленный по месту. Оператор области действия имеет ограничение на указание повторений: для самого шаблона области действия допустимо только указание необязательности. При этом для шаблонов в левой и правой части оператора никаких ограничений нет.

Следование через слова и разделители.

Поиск определённых слов и конструкций, находящихся на некотором расстоянии в тексте друг от

друга, является популярной задачей. В языке описания шаблонов для этого существует соответствующий оператор следования через слова. В операторе следования через слова под словом понимается стандартный шаблон Word. Расстояние между левой и правой частью задаётся в словах и записывается аналогично повторениям, но без квадратных скобок. В случае нескольких пересекающихся вариантов совпадения результирующим считается совпадение меньшей длины. Это связано с тем, что при поиске связанных по смыслу слов, чем меньше расстояние между словами, тем вероятнее они связаны и являются целью поиска. Оператор следования через слова также допускает указание шаблона, совпадение которого между левой и правой частью недопустимо. Если необходимо запретить несколько конструкций, то следует использовать вариацию. В общем виде оператор следования через слова записывается следующим образом:

```
X .. M-N ~Z .. Y
```

Здесь определено следование Y за X на расстоянии от M до N слов, причём между X и Y не должно находиться шаблона Z. Эквивалентную данной конструкцию можно также выразить через уже описанные операторы и стандартные шаблоны

```
X + [M-N](WordBreaks + {Word, ~X, ~Y, ~Z}) +
?{WordBreaks, ~Y} + Y
```

Важной частью приведённого примера является включение шаблонов X и Y в виде исключений вариации. Это обеспечивает выполнение требования о выборе самого короткого совпадения из пересекающихся: между X и Y не может находиться ни X, ни Y. Частным случаем следования через слова является следование через разделители, т.е. следование через ноль слов. Сокращённая форма записи такой конструкции представлена шаблоном

```
X ... Y
```

который также может быть выражен эквивалентной цепочкой

```
X + ?{WordBreaks, ~Y} + Y
```

Язык описания шаблонов не специфицирует способ реализации данного оператора на уровне исполнителя поиска: допустимо как выражение через другие операторы, так и отдельная реализация.

Последовательное упоминание.

В некоторых случаях требуется найти некоторые слова в тексте, наподобие того, как это выполняется при полнотекстовом поиске. Для этого в языке описания шаблонов используется оператор последовательного упоминания, обозначаемый символом "&". Считается, что шаблоны последовательно упоминаются в тексте, если они встречаются в нем на любом расстоянии и в любой последовательности. Оператор последовательного упоминания можно выразить через другие операторы, например, шаблон

```
X & Y
```

можно заменить на эквивалентный шаблон

```
{X .. 0+ .. Y, Y .. 0+ .. X}
```

Язык описания шаблонов не специфицирует способ реализации данного оператора на уровне исполнителя поиска, допустимо как выражение оператора через другие операторы, так и отдельная реализация.

Приоритет операций.

В следующей таблице перечислены приоритеты операций в языке описания шаблонов в порядке убывания приоритета.

Приоритет	Название	Обозначение
1	Исключение элемента вариации	~
2	Повторение	[]
3	Вариация	{ }
4	Последовательное упоминание	&
5	Последовательность	+
6	Следование через слова и разделители	..
7	Область действия	@

В случае необходимости приоритет операции можно задать явно с помощью круглых скобок. Например, в шаблоне

```
[1-5] (X + Y)
```

повторение относится ко всей цепочке. Без круглых скобок повторение будет относиться только к шаблону X.

Именованные шаблоны и теги.

Если необходимо сделать ссылку на шаблон из другого шаблона или вернуть его в результатах поиска, то ему следует задать уникальное имя. Имя шаблона представляет собой идентификатор, т.е. начинается с буквы или знака подчёркивания, за которым следует последовательность букв, цифр и знаков

подчёркивания. Такой шаблон называется именованным. За именем через знак равенства следует сам шаблон. Определение именованного шаблона завершается точкой с запятой. Например, в шаблоне

```
Z = X;
```

именованный шаблон Z определён через шаблон X. Совпадение X означает совпадение Z. Идентификатор Z может использоваться для ссылки на шаблон в других шаблонах. Для того чтобы шаблон возвращался в результатах поиска, перед шаблоном следует поставить знак "#". Помеченный таким образом шаблон называется целевым шаблоном или тегом. Например, для определения тега Z через цепочку X и Y используется шаблон

```
#Z = X + Y;
```

Именованные шаблоны, не являющиеся тегами, используются для создания других шаблонов, но в результат поиска они не попадают.

Правила обработки текста.

Правила разбора входного текста на лексемы.

Разбор входного текста на лексемы реализован на основе следующих правил:

а) начало и конец текста должны выделяться как лексемы нулевой длины с типами Start и End соответственно;

б) последовательность из символа возврата каретки и символа перевода строки должна выделяться как лексема с типом NewLine;

в) последовательность буквенных символов должна выделяться как лексема с типом Alpha;

г) последовательность цифровых символов должна выделяться как лексема с типом Num;

д) последовательность цифровых и буквенных символов, начинающаяся с буквы, должна выделяться как лексема с типом AlphaNum;

е) последовательность цифровых и буквенных символов, начинающаяся с цифры, должна выделяться как лексема с типом NumAlpha;

ж) последовательность пробельных символов должна выделяться как лексема с типом Space;

з) каждый знак препинания должен выделяться как отдельная лексема с типом Punct;

и) каждый небуквенный знак, не относящийся к знакам препинания, должен выделяться как отдельная лексема с типом Symbol.

Правила трансляции шаблонов поиска.

Трансляция шаблонов поиска с языка описания шаблонов должна быть реализована с учётом следующих правил:

а) текстовое описание шаблонов должно транслироваться в дерево выражений, используемое исполнителем поиска;

б) попытка трансляции некорректного текстового описания шаблонов должна приводить к формированию программного исключения с информацией об ошибке;

в) текст описания шаблонов для трансляции должен загружаться из файла или браться из текстовой строки;

г) текстовые литералы, состоящие более чем из одной лексемы, должны преобразовываться в последовательности лексем на основе тех же правил лексического разбора, что используются для входного текста;

д) в процессе трансляции текстового описания шаблонов может выполняться замена одних выражений в дереве выражений на другие эквивалентные выражения с целью оптимизации используемой памяти и скорости поиска.

Общие правила поиска.

Алгоритм поиска должен быть реализован с учётом следующих правил:

а) поиск множества шаблонов в одном документе должен выполняться за один просмотр последовательности лексем, соответствующих документу;

б) в случае обнаружения нескольких пересекающихся совпадений одного и того же шаблона, результирующим должно считаться наибольшее по длине в лексемах совпадение, которое началось раньше по тексту;

в) максимальный используемый объем памяти при поиске должен ограничиваться максимальным количеством одновременно рассматриваемых вариантов совпадений;

г) результаты поиска должны быть представимы в иерархическом виде, где для каждого элемента шаблона хранится участок текста, с которым найдено совпадение, и аналогичные результаты для его вложенных элементов;

д) результаты поиска должны быть представимы в виде списка, где сохранены только совпадения тегов, без детализации вложенных элементов;

е) в качестве результатов поиска должны возвращаться только совпадения тегов, т.е. совпадения шаблонов, помеченных знаком "#".

Правила поиска текстовых литералов и типов лексем.

Поиск текстовых литералов и типов лексем должен быть реализован с учётом следующих правил:

а) поиск текстовых литералов должен выполняться в соответствии с заданной чувствительностью к регистру символов;

б) поиск типа лексемы должен выполняться в соответствии с типом лексемы, присвоенным на этапе разбора входного текста на лексемы, без дополнительного анализа текста лексемы.

Правила поиска последовательностей.

Поиск последовательностей должен быть реализован с учётом следующих правил:

а) в качестве элемента последовательности может быть задан любой шаблон;

б) последовательность должна считаться совпавшей, когда совпали все её обязательные элементы;

в) при поиске необязательных элементов последовательности должны всегда рассматриваться оба варианта совпадения последовательности: с необязательным элементом и без, даже в том случае, когда совпадение необязательного элемента найдено;

г) последовательность, все элементы которой являются необязательными, должна считаться необязательной.

Правила поиска вариаций.

Поиск вариаций должен быть реализован с учётом следующих правил:

а) в качестве альтернативы может быть задан любой шаблон;

б) в качестве исключения может быть задан любой шаблон;

в) при поиске совпадения вариации должны независимо рассматриваться варианты совпадения каждой из её альтернатив;

г) совпадение шаблона-исключения должно отменять совпадение только тех шаблонов-альтернатив, которые совпали или начали совпадать с той же начальной позиции, что и шаблон-исключение;

д) вариация, хотя бы один из элементов-альтернатив которой является необязательным, должна считаться необязательной.

Правила поиска повторений.

Поиск повторений должен быть реализован с учётом следующих правил:

а) повторение может быть задано без ограничений для любого шаблона, кроме шаблона области действия;

б) для шаблона области действия кроме значения повторения по умолчанию должно быть доступно только указание повторения от нуля до единицы, другие значения должны быть запрещены;

в) шаблон должен считаться повторяющимся, если он совпадает последовательно несколько раз, причём следующее совпадение начинается с лексемы, следующей за той, которой оканчивается предыдущее повторение;

г) должны рассматриваться все варианты совпадения элемента с заданным в виде диапазона числом повторений;

д) элементы с минимальным числом повторений, равным нулю, должны считаться необязательными;

е) для элементов, у которых не задано максимальное число повторений, это число должно задаваться таким образом, чтобы время поиска совпадений не превышало интервал ожидания ответа вызывающей системой.

Правила поиска шаблонов с областью действия.

Поиск шаблонов с областью действия должен быть реализован с учётом следующих правил:

а) в качестве левого операнда может быть задан любой шаблон;

б) в качестве правого операнда может быть задано как имя шаблона, так и объявленный по месту шаблон;

в) шаблон должен считаться совпавшим в области действия другого, если левая граница совпадения первого шаблона находится не раньше в тексте, чем левая граница совпадения второго шаблона, а правая не позже в тексте, чем правая граница совпадения второго шаблона.

Правила поиска шаблонов, следующих через слова и разделители.

Поиск шаблонов, следующих через слова и разделители, должен быть реализован с учётом следующих правил:

а) в качестве операндов могут быть заданы любые шаблоны;

б) при подсчёте числа слов под словом должна пониматься цифровая, буквенная или буквенно-цифровая лексема, разделители при подсчёте должны игнорироваться;

в) в случае нескольких пересекающихся вариантов совпадения результирующим должно быть совпадение с меньшим расстоянием в словах между шаблонами;

г) совпадение шаблона-исключения должно приводить к отмене совпадения шаблона следования через слова;

д) в операторе следования через слова и разделители под разделителем должна пониматься непустая последовательность пробельных символов, знаков пунктуации, переводов строки и других небуквенных знаков.

Правила поиска последовательно упоминающихся шаблонов.

Поиск последовательно упоминающихся шаблонов должен быть реализован с учётом следующих правил:

а) в качестве операндов могут быть заданы любые шаблоны;

б) под последовательно упоминающимися шаблонами должны пониматься шаблоны, которые встречаются в тексте на любом расстоянии и в любом порядке.

Правила поиска ссылок на шаблоны.

Поиск ссылок на шаблоны должен быть реализован с учётом следующих правил:

а) в случае, когда некоторый участок текста совпадает с шаблоном, на который ссылается несколько других шаблонов, поиск шаблона, который находится по ссылке, должен выполняться один раз;

б) должен выполняться поиск шаблонов, структура ссылок между которыми образует левую или правую рекурсию;

в) при ссылке на стандартный шаблон выражение, соответствующее стандартному шаблону, должно напрямую подставляться в месте ссылки.

Разбор входного текста на лексемы.

Частью данного изобретения является один из вариантов лексического анализатора текста 19, который выполняет разбор входного текста на лексемы. Входными данными для лексического анализатора является текстовая строка. Иных данных для работы не требуется, так как правила разделения на лексемы зафиксированы. Под текстовой строкой в данном случае понимается объект строки в языке программирования, логически представляющий собой массив символов в оперативной памяти. Решение о работе с данными только в оперативной памяти связано с необходимостью высокой производительности данного этапа поиска. Загрузка текстовой строки в память увеличивает вероятность кэширования как частей самой строки, так и программного кода обращения к символу, который при таком подходе является простым обращением по индексу в массиве. Таким образом, разрешение компромисса между затратами времени и памяти в данном случае выполняется в пользу времени. Стоит отметить, что такое же решение принято и в алгоритме поиска: выявление многих шаблонов в тексте за один просмотр текста требует дополнительный объем памяти для хранения состояния всех совпадений. При этом следует учитывать, что в языках с автоматическим управлением памятью увеличение потребления памяти за счёт увеличения числа создаваемых объектов косвенно оказывает влияние на производительность, так как растут накладные расходы на выполнение сборки мусора. В связи с этим выходными данными лексического анализатора является специальный индекс, содержащий исходную строку текста и массив структур, кодирующих тип и границы каждой лексемы в исходном тексте. Создание подстроки для каждой лексемы из исходного текста выполняется по мере необходимости. Такая структура резко сокращает количество объектов, создаваемых в динамической памяти, по сравнению с простым созданием новой подстроки для каждой лексемы.

Лексический анализатор использует правила деления на слова стандарта Unicode Standard Annex #29 со следующими модификациями:

символ точки и знак подчёркивания разделяют буквенную или буквенно-цифровую последовательность на несколько лексем;

десятичный разделитель разделяет цифровую последовательность на несколько лексем;

последовательность пробельных символов не разделяется;

все знаки препинания и другие небуквенные знаки выделяются отдельной лексемой.

Для применения правил определения границы слова используются свойства границы слова. Эти свойства определены для всего диапазона символов Unicode, в том числе для тех, размер которых превышает два байта. В данном решении применяется формат UTF-16, предполагающий использование двух байтов для одного символа. Символы, выходящие за два байта, относятся к древним или редко используемым языкам, поэтому игнорируются.

В процессе обработки текста требуется определять свойство границы слова для каждого символа, выполняя поиск в некоторой структуре данных, в которой хранится информация о свойстве границы слова для каждого двухбайтового символа Unicode. Для эффективного решения данной задачи используется статический двумерный массив свойств границы слова. Адресация осуществляется по старшему и младшему байту символа, для которого требуется получить свойство границы слова. Последовательность свойств символов имеет следующие характеристики: наиболее часто встречаются непрерывные последовательности значений "любой" и "буква алфавита". Для сокращения размера массива для таких последовательностей на втором уровне адресации используются не массивы с одинаковыми значениями свойства для каждого символа, а ссылки на пустые массивы-маркеры. Таким образом, при поиске требуется сначала выделить старший байт искомого символа и использовать его для первого уровня адресации в массиве. Ссылку на полученный массив следует сначала сравнить с двумя массивами-маркерами. В случае совпадения с одним из них тип символа определён. Если же совпадения не найдено, то необходимо выделить младший байт символа и использовать его для адресации в полученном массиве.

Кроме выявления границ лексический анализатор определяет также тип лексем. Для выявления типов лексем для каждого нового символа отслеживается, к какому типу он относится, а также какой тип имеет лексема с учётом рассмотренных ранее символов. Символы разделяют на буквы, цифры, знаки

пунктуации, знак возврата каретки и перевода строки. Прочие символы считаются небуквенными знаками. Данная логическая модель является конечным автоматом, где состоянием является тип лексемы с учётом рассмотренных ранее символов, а переход между состояниями осуществляется по входному символу. Сброс автомата в исходное состояние, когда тип лексемы считается неопределённым, выполняется на границах лексем. Главный цикл лексического анализатора, в котором выполняется просмотр строки, координирует действия разделителя на лексемы и классификатора лексем. Полный перечень правил определения границы лексемы предполагает буфер свойств символов размером четыре элемента. Кроме текущего символа используется просмотр вперёд на два символа и назад на один. Переход к следующему символу предполагает определение для него свойства границы слова и добавление значения свойства в буфер. В начале обработки текста для инициализации буфера выполняется переход на два символа вперёд. Далее в главном цикле происходит движение вперёд по строке, в процессе которого происходит продвижение буфера и подача символов в классификатор лексем. После обработки каждого символа выполняется проверка, нет ли границы лексемы в текущей позиции. В случае, когда на данном символе лексема завершилась, её границы и тип добавляется в результат. Также в соответствующие позиции результата добавляются искусственные лексемы начала и конца текста нулевой длины.

Представление шаблонов на уровне исполнителя поиска.

На уровне исполнителя поиска каждый шаблон представлен структурой данных, которая кодирует дерево всех элементов грамматики, из которых состоит шаблон. Эта структура данных называется деревом поисковых выражений и отличается от синтаксического дерева, полученного в результате трансляции шаблона, тем, что приспособлена для эффективной реализации поиска. Дерево поисковых выражений содержит лишь базовые операторы языка описания шаблонов, такие как "последовательность" и "вариация", и не содержит производные операторы, выразимые через базовые операторы, такие как "следование через слова и разделители" и "последовательное упоминание". Разделение между деревом поисковых выражений и синтаксическим деревом шаблона обеспечивает необходимую гибкость при модернизации и совершенствовании решения, позволяет оптимизировать внутренние структуры данных и алгоритмы исполнителя поиска, работающего на уровне поисковых выражений, и сохраняет совместимость с существующими клиентами, работающими на уровне синтаксического дерева.

Дерево поисковых выражений состоит из элементов (выражений) следующего типа:

лексема (класс TokenExpression);

ссылка на шаблон (класс ReferenceExpression);

последовательность (класс SequenceExpression);

вариация (класс VariationExpression);

элемент вариации (класс VariationItemExpression);

область действия (класс EnclosureExpression);

шаблон (класс PatternExpression).

Оператор повторения языка текстового описания шаблонов представлен в исполнителе поиска не отдельным типом выражения, а атрибутом других выражений. Данный атрибут определён у следующих выражений: "лексема", "ссылка на шаблон", "последовательность", "вариация" и "области действия". Такой подход не является обязательным, он выбран, чтобы сократить количество объектов, создаваемых динамически, и чтобы уменьшить глубину дерева выражений. Представление оператора повторения в виде отдельного типа поискового выражения не меняет сути изобретения.

Выражение с типом "шаблон" является корнем дерева поисковых выражений. Подчинённым элементом этого выражения может быть выражение любого из вышеперечисленных типов, кроме типов "шаблон" и "элемент вариации". Выражения с типом "лексема" и "ссылка на шаблон" являются листовыми (терминальными) узлами в дереве, т.е. не могут иметь подчинённых элементов; все остальные выражения являются составными. В терминах ООП классы терминальных выражений (классы TokenExpression и ReferenceExpression) являются производными от класса TerminalExpression. Классы составных выражений (классы SequenceExpression, VariationExpression, VariationItemExpression, EnclosureExpression и PatternExpression) могут включать в себя подчинённые выражения и являются производными от класса CompoundExpression. Базовым классом для классов TerminalExpression и CompoundExpression, а косвенно для всех классов выражений, является класс Expression. Все поисковые выражения с типом "шаблон" собраны в пакет шаблонов.

Пример шаблона и соответствующего ему дерева выражений приведён на фиг. 2. Как принято в информатике, дерево рисуется перевёрнутым: корень находится вверху, а терминальные элементы - внизу.

Дерево поисковых выражений каждого шаблона является полностью связным, позволяя не только спускаться от корня вниз к терминальным элементам, но и подниматься от терминальных элементов к головным элементам вверх по дереву. Это важно для работы исполнителя поиска.

Каждое поисковое выражение содержит поисковый индекс. Поисковый индекс - это структура данных (объект), который позволяет по заданной входной лексеме или по заданному совпавшему шаблону быстро получить терминальные элементы (выражения с типом "лексема" и "ссылка на шаблон") всех подчинённых выражений, с которых может начинаться совпадение или несовпадение головного выраже-

ния. Поисковый индекс головного выражения есть результат слияния поисковых индексов подчинённых выражений, причём способ слияния зависит от типа головного выражения. Если головное выражение - это вариация, то её поисковый индекс - это объединение, в смысле объединения множеств, поисковых индексов всех элементов вариации. Если головное выражение - это последовательность, то её поисковый индекс - это объединение, в смысле объединения множеств, поисковых индексов всех первых необязательных элементов с поисковым индексом первого обязательного элемента последовательности. Необязательным считается элемент, у которого минимально допустимое число повторений равно нулю. Для терминального выражения (лексема или ссылка на шаблон) поисковый индекс состоит, по сути, из одного элемента и возвращает в результатах поиска либо один этот элемент, либо пустое множество. Очевидно, что полное или частичное включение поисковых индексов подчинённых элементов в поисковые индексы головных элементов требует заметных затрат оперативной памяти, поэтому целесообразно проводить оптимизацию с целью сокращения дублирования одних и тех же данных, в частности текстовых строк, в памяти.

Поисковый индекс должен обеспечивать разные виды быстрого поиска выражений: только лексем, только ссылок на шаблоны, любых терминальных выражений. Причём критериями поиска могут быть значения типов лексем и текстовых литералов с учётом и без учёта регистра символов, идентификаторы шаблонов, признаки учёта исключений в вариациях, т.е. терминальных выражений, приводящих к совпадению или не совпадению поискового выражения.

Поисковые индексы всех корневых поисковых выражений с типом "шаблон" объединяются, в смысле объединения множеств, в корневой поисковый индекс. Корневой поисковый индекс является частью пакета шаблонов.

В терминах ООП поисковые выражения представляют собой объекты соответствующих классов и имеют следующие основные свойства и процедуры:

ParentExpression - ссылка на головное выражение;

RepeatRange - структура, в которой хранится допустимое число повторений;

IsRepeatable - определяет, допустим ли оператор повторения для данного выражения (переопределяется в производных классах);

IsOptional - определяет, является ли выражение необязательным;

GetOrCreateExpressionIndex - процедура для получения поискового индекса выражения; если доступ происходит впервые, поисковый индекс будет предварительно создан;

CreateExpressionIndex - процедура, переопределяемая в производных классах, которая непосредственно выполняет построение индекса для соответствующего типа выражения.

После полного создания пакета шаблонов с поисковыми выражениями и поисковыми индексами, эти структуры данных больше не меняются, в процессе работы исполнителя поиска доступ к ним выполняется только по чтению.

Трансляция текстового описания шаблонов.

Язык описания шаблонов, предложенный в рассматриваемом варианте осуществления изобретения, отличается простым и понятным синтаксисом, который к тому же удобен для автоматического разбора. Трансляция описания шаблонов с этого языка в поисковые выражения выполняется с помощью хорошо известных рекурсивных алгоритмов с использованием машинного или программного стека.

В рассматриваемом варианте реализации изобретения трансляция текстового описания шаблонов состоит из двух этапов: построение и оптимизация синтаксического дерева по текстовому описанию шаблонов и генерация поисковых выражений с поисковыми индексами для исполнителя поиска.

Построение синтаксического дерева выполняется за один просмотр лексем языка шаблонов с помощью рекурсивного алгоритма и с использованием машинного стека. После выделения следующей лексемы её текстовое представление и тип сохраняются в состоянии синтаксического анализатора. Типы лексем в данном случае отличаются от таковых в лексическом анализаторе текста, так как в данном случае идёт разбор не естественного, а формального языка. Выделяются идентификаторы, строковый и числовой литералы, круглые, фигурные и квадратные скобки, арифметические знаки, а также другие специальные символы, имеющие особый смысл в языке описания шаблонов.

Синтаксический разбор формальных грамматик выполняется методом восходящей рекурсии. Конечный результат разбора текста на языке описания шаблонов - это объект пакета шаблонов, представленный в виде синтаксического дерева. Разбор начинается с вызова процедуры разбора пакета, который в цикле вызывает процедуру разбора синтаксиса именованного шаблона. Процедура разбора синтаксиса именованного шаблона после определения имени шаблона выполняет процедуру разбора правой части - тела шаблона. В этой процедуре выполняется вызов процедуры разбора самого низкоприоритетного оператора - оператора области действия. В дальнейшем в процедуре разбора синтаксиса оператора области действия вызывается процедура разбора предыдущего по приоритету оператора - оператора следования через слова и разделители, и так далее. В местах, где может быть записано любое выражение, происходит вызов процедуры разбора главного выражения, в котором последовательность вызовов начинается сначала, а также обрабатывается указание требуемого приоритета операторов с помощью круглых скобок. Такой подход позволяет при разборе получить требуемый приоритет операторов. Полученное в ре-

зультате разбора синтаксическое дерево содержит узлы, которые точно соответствуют операторам языка описания шаблонов.

Следующим шагом является преобразование и оптимизация синтаксического дерева и связывание шаблонов. Под связыванием шаблонов понимается запись ссылок на синтаксические деревья именованных шаблонов в местах ссылок на них. Выполнение этого шага в конце позволяет не беспокоиться о порядке записи шаблонов, использовать ссылки вперёд на шаблоны, объявленные позже, а также создавать рекурсивные шаблоны. На этом шаге выполняются следующие преобразования и оптимизации:

замена операторов следования через слова и разделители и операторов последовательного упоминания на эквивалентные конструкции из базовых операторов;

лексический анализ текстовых литералов и замена на эквивалентную последовательность лексем, если текстовый литерал состоит более чем из одной лексемы;

замена последовательностей и вариаций, состоящих из одного элемента, на сам элемент;

если положительный элемент вариации сам является вариацией, то его элементы будут добавлены в родительскую вариацию напрямую.

Обход и обновление синтаксического дерева реализуется с использованием приёма проектирования "посетитель", называемого также двойной диспетчеризацией. В процессе обхода синтаксического дерева с помощью объекта "посетителя" в случае обновления какого-либо узла происходит рекурсивное перестроение всех его родительских узлов. Для удовлетворения правилам синтаксического разбора текстовых литералов они разделены на два вида: лексемы и составные текстовые литералы. В процессе разбора текста синтаксический анализатор считает все текстовые литералы составными, а при обходе дерева каждый такой литерал заменяется либо на одну лексему, либо на последовательность из лексем.

Генерация дерева поисковых выражений из синтаксического дерева шаблонов тоже выполняется через обход дерева с помощью приёма проектирования "посетитель". В процессе обхода синтаксического дерева используется стек создаваемых выражений. Происходит рекурсивный спуск вглубь до терминальных элементов и создание для них выражений, а затем на подъёме из рекурсии создаются выражения для всех родительских элементов. Особым образом происходит создание выражений для ссылок на шаблоны и для оператора области действия. В обоих случая требуется дополнительно выполнять связывание ссылок после создания всех выражений. В случае с оператором области действия дополнительной особенностью является оптимизация поиска: если в правой части оператора находится шаблон, объявленный по месту, то для выполнения поиска требуется создать анонимный шаблон и поместить в его правую часть записанное по месту выражение. Ссылка на полученный анонимный шаблон используется при конструировании выражения для оператора области действия. Однако если в правой части оператора сразу записана ссылка на шаблон, то необходимо использовать сразу её и не создавать анонимный шаблон.

На всех этапах трансляции в случае любых ошибок происходит создание программного исключения с информацией об ошибке. Этот подход используется для обнаружения и обработки ошибок синтаксиса, ошибок входных данных, а также внутренних ошибок, в частности, недопустимых состояний синтаксического анализатора или генератора выражений.

Конечным результатом трансляции шаблонов является пакет шаблонов, содержащий поисковые выражения с собственными поисковыми индексами, а также корневой поисковый индекс.

Структуры данных кандидатов и результатов совпадений.

Проверка совпадения всех шаблонов из пакета шаблонов выполняется за один последовательный просмотр лексем текста. Для реализации этой возможности исполнитель поиска хранит и использует на каждом шаге просмотра данные о состоянии проверки поисковых выражений, из которых состоят шаблоны. Структура данных, которая хранит информацию о полном или неполном совпадении текста с поисковым выражением, начиная с некоторой позиции в тексте, и служит для принятия решения о дальнейшем совпадении или несовпадении фрагмента текста с поисковым выражением, называется кандидатом совпадения (далее просто "кандидат"). Как и поисковые выражения, кандидаты совпадений образуют дерево кандидатов, где корнем дерева является кандидат совпадения всего шаблона, а элементами дерева являются кандидаты совпадения для последовательностей, элементов вариаций, лексем и других типов поисковых выражений. Дерево кандидатов каждого шаблона является полностью связным, позволяя не только спускаться от корня вниз к терминальным элементам, но и подниматься от терминальных элементов к головным элементам вверх по дереву, что важно для работы исполнителя поиска. Множество всех кандидатов всех шаблонов, проверяемых на каждом шаге просмотра лексем текста, называется набором кандидатов и образует состояние исполнителя поиска.

По мере просмотра лексем текста кандидаты совпадений создаются в наборе кандидатов, затем используются для принятия решения о совпадении или несовпадении шаблона с читаемыми лексемами текста, и в конце концов удаляются из набора кандидатов либо по причине несовпадения с очередной лексемой, либо по причине превращения кандидатов в результаты совпадения. Таким образом, результат совпадения фрагмента текста с шаблоном - это кандидат совпадения шаблона, по которому окончательно принято решение о совпадении, и который содержит исчерпывающую информацию, объясняющую, где и почему произошло совпадение. Результаты совпадения шаблонов используются для принятия решения

о совпадении кандидатов ссылок на эти шаблоны. Результаты совпадения целевых шаблонов (тегов) возвращаются клиенту исполнителя поиска, причём по требованию клиента результаты возвращаются либо в полном, либо в сокращённом виде. В последнем случае это могут быть лишь позиции начальной и конечной лексем текста, соответствующих совпадению с шаблоном. Дерево кандидатов состоит из элементов (кандидатов) следующего типа:

лексема (класс TokenCandidate);

ссылка на шаблон (класс ReferenceCandidate);

последовательность (класс SequenceCandidate):

элемент вариации (класс VariationItemCandidate) подразделяется на кандидата альтернативы (класс Inclusive VariationItemCandidate) и кандидата исключения (класс ExclusiveVariationItemCandidate);

область действия (класс EnclosureCandidate);

вхождение шаблона (класс PatternCandidate).

Для кодирования повторений дополнительного типа кандидата не требуется. Каждое повторение представляется отдельным кандидатом, в котором указан порядковый номер повторения. Кандидат хранит номер своего повторения, а родительский кандидат хранит ссылки на кандидатов всех повторений своих подчинённых элементов.

Кандидат с типом "вхождение шаблона" является корнем дерева кандидатов. Его подчинёнными элементами могут быть кандидаты других типов, кроме типа "вхождение шаблона". Кандидаты с типом "лексема" и "ссылка на шаблон" являются листовыми (терминальными) элементами в дереве, т.е. не могут иметь подчинённых элементов; все остальные кандидаты являются составными. В терминах ООП классы терминальных кандидатов (классы TokenCandidate и ReferenceCandidate) являются производными от класса TerminalCandidate. Классы составных кандидатов (классы SequenceCandidate, VariationItemCandidate, EnclosureCandidate и PatternCandidate) могут включать в себя подчинённых кандидатов и являются производными от класса CompoundCandidate. Базовым классом для классов TerminalCandidate и CompoundCandidate, а косвенно для всех классов кандидатов является класс Candidate.

В терминах ООП кандидаты представляют собой объекты соответствующих классов и имеют следующие основные свойства:

ParentCandidate - ссылка на родительского кандидата;

RepeatNumber - номер повторения;

StartTokenNumber - номер лексемы, с которой началось совпадение кандидата;

Expression - ссылка на выражение, соответствующее кандидату.

ChildRepeatsPerPosition - двумерный список подчинённых кандидатов, где для первого уровня адресации используется номер позиции подчинённого кандидата внутри головного, а для второго уровня адресации - номер повторения; свойство определено только для составных кандидатов.

Набор кандидатов исполнителя поиска устроен особым образом, он хранит лишь кандидатов последних совпавших терминальных элементов (лексем и ссылок на шаблоны). Причём каждый вариант совпадения шаблона, начинающийся с некоторой позиции в тексте, представлен полностью обособленным деревом кандидатов. Дерево кандидатов для каждого варианта совпадения текста с шаблоном зацеплено за набор кандидатов последним совпавшим терминальным элементом. Если зацепление пропадает в результате изъятия терминального элемента из набора кандидатов, то это означает, что всё дерево кандидатов выпадает из набора кандидатов, и если это произошло не в результате полного совпадения, то дерево кандидатов превращается в "мусор", который должен утилизироваться сборщиком мусора. Сборка мусора в современных платформах программирования выполняется автоматически; для других платформ программирования утилизацию кандидатов, превратившихся в мусор, нужно выполнять вручную.

Алгоритм работы исполнителя поиска.

Блок-схема, поясняющая общий алгоритм работы исполнителя поиска 21, приведена на фиг. 3. В соответствие с блок-схемой последовательность лексем текста 20 передаётся в главный цикл исполнителя поиска 31-40, в котором для каждой лексемы T выполняются операции, начиная с 32 и заканчивая 39. Суть этих операций состоит в том, чтобы в зависимости от конкретного значения лексемы T принять решение о совпадении, несовпадении или необходимости дальнейшей проверки проверяемых кандидатов в наборе кандидатов 22, а также создать кандидатов для шаблонов, совпадение которых возможно начиная с лексемы T. Для этого в цикле 32-34 для каждого кандидата C в наборе кандидатов лексем выполняется шаг 33, в котором по лексеме T принимается решение по дереву кандидатов, соответствующего кандидату C. Действия на шаге 33 более подробно раскрыты на фиг. 4. После завершения цикла обработки кандидатов 32-34, на шаге 35 выполняется создание и добавление в набор кандидатов новых кандидатов, полученных на основе поиска в корневом поисковом индексе шаблонов, начинающихся с лексемы T. Создание новых кандидатов состоит из двух шагов: создание новых кандидатов лексем и создание новых кандидатов ссылок на шаблоны. Для создания кандидатов лексем выполняется обращение к поисковому индексу, который возвращает список терминальных поисковых выражений лексем, совпадающих с лексемой T. Для каждого полученного выражения происходит построение дерева кандидатов. Это осуществляется на подъёме вверх по дереву выражений. Если после создания кандидатов не был превышен лимит кандидатов, то происходит также создание кандидатов ссылок на шаблоны. Кандидаты

ссылок на шаблоны также являются терминальными кандидатами, но они хранятся в наборе кандидатов отдельно от кандидатов лексем. Это связано с тем, что для принятия решений в таких кандидатах необходима не лексема текста, а совпавший шаблон, ссылку на который и представляет данный кандидат. Создание дерева кандидатов для ссылки происходит аналогично созданию дерева кандидатов для лексемы: на подъёме вверх по дереву выражений. На шаге 36 проверяется, является ли лексема T лексемой конца текста, и если является, то выполняется шаг 37 для удаления всех кандидатов из набора кандидатов. Если лексема T - это любая другая лексема, то шаг 37 пропускается. На шаге 38 выполняется проверка, не превышен ли лимит кандидатов, и если превышен, то на шаге 39 в последовательность лексем искусственно вставляется лексема конца текста, которая начинает обрабатываться на следующей итерации цикла 31-40. После выхода из главного цикла 31-40 в данных исполнителя поиска остаются накопленными результаты совпадений 24. Эти данные должны удаляться перед поиском по другому тексту.

Общий алгоритм принятия решений.

Блок-схема, поясняющая общий алгоритм принятия решений по дереву кандидатов, соответствующего терминальному кандидату C, в зависимости от значения текущей лексемы T, приведен на фиг. 4. Входными данными 41 алгоритма являются кандидат C из набора кандидатов и лексема T в некоторой позиции текста. Кандидат C является терминальным кандидатом и может быть либо кандидатом лексемы, либо кандидатом ссылки на шаблон.

Принятие решения выполняется относительно дерева кандидатов, соответствующего кандидату C. Сначала на шаге 42 проверяется, помечен ли кандидат C удалённым, и если помечен, то все шаги принятия решения, начиная с 43 и заканчивая 51, пропускаются, а на шаге 52 происходит выход из алгоритма принятия решения. Если кандидат C не помечен удалённым, то на шаге 43 выполняется проверка, совпало ли дерево кандидатов полностью.

В соответствие с выбранной методологией ООП, процедуры принятия решений являются процедурами объектов, которыми представлены кандидаты, поэтому для принятия решения на шаге 43 лексема T передаётся процедуре принятия решений кандидата C. Если в процедуре кандидата невозможно принять решение, этот процесс переносится выше по дереву кандидатов путём вызова процедуры принятия решений у головного кандидата. Таким образом для принятия решения сначала происходит обращение к кандидату лексемы, а кандидат лексемы в случае необходимости делегирует принятие решения своему головному элементу и так далее до корневого элемента дерева - кандидата шаблона.

Если на шаге 43 дерево кандидатов полностью совпало, то на шаге 44 проверяется, не является ли соответствующий шаблон целевым, т.е. тегом, и если является, то на шаге 45 корневой кандидат этого дерева (кандидат шаблона) добавляется в набор результатов. Если соответствующий шаблон не являются тегом, то шаг 45 пропускается. После выполнения шага 44 и вне зависимости от шага 45, т.е. в случае, если дерево кандидатов полностью совпало, и вне зависимости от того, является ли соответствующий шаблон тегом, происходит принятие решения для кандидатов ссылок на этот шаблон на шаге 48. Принятие решения выполняется для кандидатов ссылок, присутствующих в наборе кандидатов, и начинающихся с той же самой позиции в тексте, с которой начинается кандидат совпавшего шаблона. Фактически шаг 48 означает рекурсивное обращение к алгоритму принятия решения на фиг. 4.

Если на шаге 43 дерево кандидатов не совпало полностью, то на шаге 46 выполняется проверка того, что дерево кандидатов продолжает совпадать, и, если нет, то на шаге 47 дерево кандидатов помечается к удалению, а на следующем шаге 48 выполняется рекурсивное обращение к алгоритму принятия решения для кандидатов ссылок на соответствующий не совпавший шаблон.

Если на шаге 46 оказывается, что дерево кандидатов продолжает совпадать, то на шаге 49 выполняется проверка, существует ли более одного варианта совпадения. Если вариантов совпадения несколько, то на шаге 50 создаются логические копии дерева кандидатов для каждого варианта совпадения, причём логические копии совпадают в той части выражения, которая предшествует лексеме T, и отличаются в той части выражения, которая соответствует разным вариантам совпадения лексемы T с лексемами в поисковых выражениях шаблона. Если существует всего один вариант совпадения, то шаг 50 пропускается, копии дерева кандидатов не создаются, а используется имеющееся дерево кандидатов. На следующем шаге 51 для каждого варианта совпадения в дереве кандидатов добавляется кандидат лексемы T, который в свою очередь добавляется в набор кандидатов.

Правила принятия решений.

Правила принятия решений по полному совпадению, несовпадению или частичному совпадению кандидатов зависят от типов кандидатов.

Общей частью правил принятия решения во всех кандидатах является анализ номера текущего повторения. Если повторение еще не достигло минимально допустимой границы, то необходимо проверить, может ли начаться совпадение нового повторения шаблона со следующей лексемы текста, и если может, то создать кандидата нового повторения. В случае, когда совпадение нового повторения невозможно, головному кандидату передаётся информация о несовпадении его подчинённого элемента. Эта информация передается выше по дереву до его корня. В случае, когда повторение достигло максимально допустимой границы, в текущем кандидате принимается решение о совпадении, которое сразу передаётся его головному кандидату. Самым сложным случаем является совпадение кандидата с номером повто-

рения, который находится между минимальной и максимальной границей. При совпадении элемента с вариативным числом повторений, необходимо рассмотреть два варианта: когда данное совпадение было последним и когда будет выполняться поиск еще одного совпадения. Приведенные варианты по результату проверки могут образовывать различные деревья кандидатов, что в итоге может привести к различным совпадениям соответствующего шаблона. Для того чтобы учитывать все варианты, необходимо выполнить копирование дерева кандидатов вплоть до корневого элемента - кандидата шаблона. После копирования один экземпляр кандидата руководствуется правилами принятия решения при достижении максимального числа повторений, а второй - правилами принятия решения в случае, когда повторение еще не достигло минимально допустимой границы.

К логике принятия решения на основе номера повторения в последовательности добавляется анализ номера позиции совпавшего подчинённого элемента. Последовательность считается совпавшей, когда совпала ее обязательная часть. Под обязательной частью последовательности понимается её подпоследовательность от первого до последнего обязательного элемента. Например, в последовательности

```
?A + B + ?C + D + ?E
```

обязательной частью является подпоследовательность от B до D с необязательным элементом C в середине.

В случае, если после обязательной части следуют необязательные элементы, необходимо также проверить, возможно ли их совпадение. Для этого в момент совпадения обязательной части последовательности, как и в случае с вариативным значением повторения, происходит копирование дерева кандидатов. В одном из экземпляров выполняется попытка продолжить совпадение необязательной части последовательности, а другой экземпляр считается совпавшим.

Для вариации правила принятия решений также имеют свои особенности. Это связано с логикой работы исключений: совпадение исключения должно отменить совпадение всех альтернатив, совпадение которых началось с той же позиции в тексте.

При этом такая отмена может произойти как до совпадения кандидата альтернативы, так и после, спустя некоторое количество лексем текста. При этом основной сценарий использования для предварительного просмотра вперед предполагает второй случай. Более сложным сценарием является отмена с учетом повторений элементов. Например, если исключение совпадает с позиции некоторого не первого повторения элемента, то отменяется совпадение только тех элементов, позиции в тексте которых больше позиции совпадения, к которому относится исключение. Для связи между деревьями кандидатов исключений и альтернатив используется состояние совпадения вариации (далее просто "состояние вариации"). Состояние вариации хранит ссылки на всех кандидатов в специальной структуре, удобной для поиска по позиции первой совпавшей лексемы. Состояние вариации связано с объектом поискового выражения вариации и позицией лексемы, на которой был создан первый кандидат элемента вариации. Все кандидаты элементов одной вариации, созданные во время обработки одной лексемы, связываются одним состоянием вариации. В процессе принятия решения в кандидате происходит обращение к состоянию вариации для получения дополнительной информации и обновления состояния.

Из-за того, что выявление всех шаблонов осуществляется за один просмотр текста, проверка исключений в вариациях должна проходить одновременно с поиском совпадений следующих за вариацией элементов. При этом совпавшие кандидаты альтернатив, для которых еще ведётся проверка исключений, считаются совпавшими неокончательно. Свойство неокончательного совпадения распространяется и на головных кандидатов. Так, например, если все элементы последовательности совпали, но хотя бы один из них на любой позиции совпал неокончательно, то и вся последовательность считается совпавшей неокончательно.

Примером работы описанного механизма является поиск шаблона

```
#P = {A, ~(A + B)}
```

во входной последовательности лексем "AAB". Здесь и далее в примерах буквами латинского алфавита обозначаются лексемы поисковых выражений и соответствующие им лексемы входного текста. Приведенный шаблон выявляет лексему A, за которой не следует лексема B.

Пошаговое изменение состояния исполнителя поиска приведено на фиг. 5. Для простоты на фиг. 5 приводится список лексем без лексемы начала текста, так как на поиск рассматриваемого шаблона она влияния не оказывает. Блоки 110, 120, 130, 140 соответствуют снимкам состояния исполнителя поиска и показывают изменение этого состояния в направлении от 110 к 140 по мере обработки лексем текста: лексемы A в позиции 0 (A_0), лексемы A в позиции 1 (A_1), лексемы B в позиции 2 (B_2) и лексемы END конца текста. На каждом из снимков 110, 120, 130, 140 показаны состояния для набора кандидатов 22 и для результатов совпадений 24. Обработка каждой лексемы состоит из двух шагов: принятие решений для кандидатов, созданных во время обработки предыдущей лексемы (далее просто "принятие решений"), и создание новых кандидатов на основе корневого поискового индекса (далее просто "создание новых кандидатов").

Во время обработки первой лексемы A_0 никаких кандидатов ранее создано не было, поэтому шаг принятие решений пропускается и происходит создание кандидатов на основе корневого индекса. По

лексеме A из корневого индекса извлекается две лексемы, относящиеся к двум элементам вариации: альтернативе и исключению. По ним создаются соответственно кандидаты 113 и 117, а также дерево кандидатов с корневым кандидатом шаблона 111. При этом элементы вариации 112 и 115 оказываются связанными с одним состоянием вариации 114. Итоговое состояние после этого шага показано на снимке 110.

При обработке второй в тексте лексемы A_1 происходит принятие решения по набору кандидатов и обнаруживается совпадение дерева кандидатов, соответствующего корневому кандидату 111 шаблона P в позиции 0. Это происходит из-за совпадения его подчинённого элемента - альтернативы вариации 112. Альтернатива вариации 112 совпадает сама по себе, так как совпадает ее подчинённый элемент 113. Но окончательное её совпадение происходит только после того, как проверка всех исключений вариации, созданных на той же позиции в тексте, завершается с отрицательным результатом. Порядок обработки кандидатов в данном случае не имеет значения. В случае, когда обработка начинается с кандидата 117, сначала будет установлено, что исключение вариации 115 не совпадает, принятие решения для альтернативы вариации 112 будет отложено, так как подчинённый кандидат самой альтернативы 113 ещё не был обработан. После обработки подчинённого кандидата альтернативы 113 будет выполнена проверка, что связанных с альтернативой исключений вариации нет, и можно принять решение об окончательном совпадении. В другом случае, когда обработка начинается с кандидата 113, будет обработан кандидат альтернативы вариации 112, в итоге будет принято решение о неокончательном совпадении, так как не завершена проверка исключений вариации. Информация о неокончательном совпадении будет передана головному кандидату 111 - кандидату шаблона P, но сохранения в результат в таком случае пока не произойдёт. Сохранение совпавшего кандидата шаблона 111 в результат произойдёт только после его окончательного совпадения в следствие обработки и принятия решения как для кандидата 113, так и для кандидата 117. В рассматриваемом сценарии это произойдет после того, как будет принято решение о несовпадении исключения вариации. Состояние вариации накапливает информацию о совпадении или несовпадении альтернатив и исключений, начавшихся с одной и той же позиции, и позволяет принять окончательное решение о совпадении связанных альтернатив вариации после того, как будут приняты окончательные решения по всем связанным исключениям вариации. В частности, состояние вариации 114 накопит информацию о совпадении альтернативы 112 и позволит принять окончательное решение о совпадении вариации, начиная с позиции 0, после того как будет принято окончательное решение о несовпадении связанного исключения 115, начинающегося с этой же позиции 0. На этапе создания новых кандидатов по второй в тексте лексеме A_1 создаётся структура, подобная той, что была создана при обработке первой лексемы A_0. Различия будут только в стартовых позициях лексем, которые хранятся в кандидатах. В частности, по лексеме A из корневого индекса извлекаются две лексемы, относящиеся к двум элементам вариации: альтернативе и исключению. По ним создаются соответственно кандидаты 123 и 127, а также дерево кандидатов с корневым кандидатом шаблона 121. При этом элементы вариации 122 и 125 оказываются связанными с одним состоянием вариации 124. Итоговое состояние после шага обработки лексемы A_1 показано на снимке 120.

Обработка следующей в тексте лексемы B_2 приведёт к созданию кандидата лексемы 138, который является вторым элементом последовательности 126 в исключении 125. Вместе с этим будет принято решение о неокончательном совпадении альтернативы 122 и всего кандидата шаблона 121, начавшегося с позиции 1. Кандидаты 123 и 127 будут удалены из набора кандидатов, а кандидат 138 будет добавлен в набор кандидатов. Дерево кандидатов, соответствующее корневому кандидату 121, на данном шаге не является "мусором", потому что оно достижимо через кандидата 138, который присутствует в наборе кандидатов. Итоговое состояние после шага обработки лексемы B_2 показано на снимке 130.

При обработке лексемы END конца текста произойдёт принятие решения по окончательному совпадению последовательности 126 и исключения вариации 125, начинающегося с позиции 1. Как следствие, принятие окончательного решения по состоянию вариации 124 приведёт к отмене совпадения всего кандидата шаблона 121. Итоговое состояние после шага обработки лексемы END конца текста показано на снимке 140.

В результате рассмотренных на фиг. 5 шагов будет, как и ожидалось, выявлено одно совпадение шаблона P в позиции 0. Ссылка на кандидата 111 данного совпадения будет сохранена в результат. Благодаря двухсторонним связям между подчинёнными и головными кандидатами из кандидата шаблона можно получить детальную информацию о совпадении всех его подчинённых элементов вплоть до совпадения терминалов с конкретными лексемами текста. В частности, по кандидату шаблона 111 можно узнать, что его совпадение обусловлено совпадением альтернативы вариации 112, которая совпала из-за совпадения кандидата 113 лексемы A с лексемой текста в позиции 0.

В приведенных правилах принятие решений осуществляется непосредственно на основе лексем текста. Поиск же ссылок на шаблоны и шаблонов в области действия основывается не непосредственно на лексемах, а на совпадении других шаблонов.

Ссылки на шаблоны, как и лексемы, являются терминальными элементами, однако кандидаты для них создаются по более сложным правилам. Они могут создаваться как другими кандидатами в процессе принятия решений, так и выбираться из корневого поискового индекса. Для ссылок два этих случая обрабатываются по-разному. В процессе принятия решений создаются кандидаты всех ссылок, которые

могут находиться на некоторой позиции. Это связано с тем, что нельзя заранее определить, возможно ли совпадение ссылки в данной позиции. Каждый кандидат ссылки после создания регистрируется в состоянии проверки ссылок, где ассоциируется с шаблоном, на который ссылается, и с позицией, в которой ожидается его совпадение.

После принятия решений выполняется создание новых кандидатов ссылок, совпадение которых уже обнаружено. Для этого используется список кандидатов тегов, совпавших во время обработки данной лексемы. Для оптимизации в такой список сохраняются не все теги, а только те, на которые есть ссылки. Эта информация сохраняется в выражении тега во время трансляции шаблонов. Из поискового индекса выбираются совпавшие ссылки и для них создаются деревья кандидатов. В отличие от констант при создании кандидата ссылки сразу происходит принятие решения на основе следующей лексемы. В процессе этого возможны случаи, когда совпадение ссылки сразу же приводит к совпадению всего шаблона. На такой шаблон в свою очередь также могут быть ссылки. Простым примером такой ситуации является рекурсивное определение

```
#P = ?P + ?A
```

Приведенный шаблон будет совпадать с любым количеством лексем A. Для поиска такого шаблона список кандидатов шаблона, совпавших во время обработки лексемы, должен пополняться во время принятия решения для ссылок, создаваемых на основе этого же списка. Для предотвращения зацикливаний для каждого нового кандидата выполняется проверка, производилась ли обработка кандидата того же шаблона, совпадение которого началось с той же позиции.

Совпадение кандидатов ссылок, которые были созданы не на основе уже совпавших шаблонов, обнаруживается благодаря состоянию проверки ссылок. После совпадения шаблона, на который возможны ссылки, выполняется поиск кандидатов ссылок, ожидающих его совпадения в заданной позиции. В случае если такие кандидаты найдены, выполняется их связывание и дальнейшее принятие решения на основе следующей лексемы.

Логика поиска шаблонов в области действия схожа с поиском ссылок. Однако в случае со ссылками начальная позиция совпадения шаблона задана строго, а область действия подразумевает совпадение одного шаблона в границах другого.

В отличие от кандидата ссылки, при создании кандидата шаблона в области действия он нигде не регистрируется. В состояние он попадает только после того, как совпадёт его дочерний элемент. Это связано с тем, что только в момент совпадения дочернего элемента становится известна позиция конца совпадения.

Для каждого тега, используемого в качестве области действия, хранится список кандидатов, ожидающих его совпадения. Данный список отсортирован по начальному и конечному номеру лексемы. Это позволяет при совпадении шаблона быстро выполнять поиск кандидатов, совпавших внутри данного шаблона. После совпадения шаблона с помощью двоичного поиска в отсортированном списке обнаруживается первый кандидат, на которого может повлиять совпадение. Начиная с данной позиции, в списке происходит проверка границ совпадения каждого кандидата и передача информации головному кандидату в случае, если найдено совпадение в области действия. Из-за того, что список отсортирован, первый найденный кандидат, находящийся за границей области действия совпавшего шаблона, останавливает просмотр списка. Совпавшие кандидаты из списка удаляются.

Рекомендации по тестированию.

Особенностью изобретения является высокая сложность положенного в его основу алгоритма. В связи с этим рекомендуется вести реализацию устройства с использованием методологии "разработка через тестирование". Разработка через тестирование предполагает сначала написание теста, покрывающего желаемую функциональность, а затем создание требуемой функциональности для прохождения созданного теста. Тесты, создаваемые в процессе такой разработки, как правило являются модульными и тестируют функции и процедуры объектов. Наличие модульных тестов значительно упрощает внесение изменений в существующий код, позволяя достаточно точно обнаруживать ошибки из-за новых изменений. Однако нужно учитывать, что для данного изобретения написание модульных тестов достаточно трудоемко. Это связано с большим количеством возможных состояний, возникающих в процессе принятия решений. В дополнение к модульным тестам рекомендуется создавать интеграционные тесты, в которых по принципу черного ящика происходит проверка соответствия фактических результатов поиска ожидаемым результатам. Для проверки рассмотренного варианта осуществления изобретения было реализовано более трехсот автоматизированных тестов, которые разделяются на следующие группы:

тесты лексического анализатора текста;

тесты синтаксического анализатора шаблонов;

тесты транслятора синтаксического дерева;

тесты выражений для представления шаблонов на уровне исполнителя поиска;

тесты кандидатов;

тесты состояния проверки ссылок;

тесты состояния проверки шаблонов в области действия;

тесты поискового индекса.

Модульные тесты разрабатывались в соответствии с принципом разработки через тестирование в процессе реализации устройства и тестируют изолированную функциональность. В модульных тестах активно применяются объекты-имитации, реализующие заданное в тестовом случае программное окружение. Наиболее трудоемким является моделирование внешнего окружения в тестах принятия решений кандидатами, однако такие тесты позволяют наиболее точно локализовать ошибки реализации.

Интеграционные тесты представляют собой тесты поиска шаблонов и представлены двумя группами: тесты, в которых шаблоны и лексемы создаются в виде объектов программы, и тесты, в которых шаблоны и текст представлены в виде строк. Такой подход необходим для возможности тестирования функций исполнителя поиска без привязки к транслятору с языка описания шаблонов. Также ручное создание поисковых выражений упрощает отладку. Для простоты при программном создании тестовых лексем не используется лексический анализатор. Вместо этого лексемы создаются вручную и представляют собой буквы латинского алфавита. Интеграционные тесты второй группы проверяют те же самые ситуации, возникающие в процессе принятия решений, однако позволяют также проверить совместную работу лексического анализатора текста, транслятора с языка описания шаблонов и исполнителя поиска на реальных данных.

Описание основных тестовых случаев приводится в следующей таблице.

Название	Шаблон	Лексемы	Совпадения
1. Лексема	A;	A	A
2. Лексема с повторением	[2] A;	AA	AA
3. Лексема с вариативным повторением	[1-2] A;	AA	AA
4. Последовательность	A + B;	AB	AB
5. Последовательность с повторением	[2] (A + B);	ABAB	ABAB
6. Последовательность с вариативным повторением	[1-2] (A + B);	ABAB	ABAB
7. Последовательность с повторяющейся вложенной последовательностью	[2-3] (A + B) + C;	ABABC	ABABC
8. Последовательность, начинающаяся с необязательного элемента	?A + B	B	B
9. Последовательность, оканчивающаяся необязательным элементом	A + ?B	A	A
10. Последовательность с необязательным элементом в середине	A + ?A + B	AB	AB
11. Последовательность с необязательным элементом в середине, содержащим исключение	A + ?{B, ~B} + B	AB	AB
12. Вложенная необязательная последовательность	C + (?A + ?B) + D	CABD	CABD
13. Вариация	{A, B}	AB	A, B
14. Вариация с повторением	[2] {A, B}	AB	AB
15. Вариация с вариативным повторением	[1-2] {A, B}	AB	AB
16. Вариация с исключением	{A, ~(A + B)}	AAB	A
17. Повторяющаяся вариация с исключением	[1-3] {C, D, ~(C + D + B)}	CCDB	C, D
18. Вложенные исключения	{A, ~{A + D, ~{A + D + C}}	ADC	A
19. Четное число одинаковых вложенных исключений	{A, ~{A, ~A}}	A	A

20. Нечетное число одинаковых вложенных исключений	{A, ~{A, ~{A, ~A}}}	A	—
21. Исключение на втором повторении	[1-2] {C, B, ~{B + D}}}	CBD	C
22. Вариация необязательных элементов в последовательности	C + {?A, ?B} + D	CD	CD
23. Отмена нескольких альтернатив	[1-3] {A, B, B + D, ~(A + B + D)}	AABD	A, BD
24. Исключение в середине последовательности	{A, ~(A + B + C + D)} + B	ABCE	AB
25. Несколько исключений разной длины	{A, ~(A + D), ~(A + B + C)}	ABCEAD	—
26. Исключение меньшей длины, чем альтернатива	{A + B + C, ~(A + D)}	ABE	—
27. Пересечение совпадений разной длины	[3] {A, B, ~(B + A + B)}	ABABAABB	A_2BA, ABB
28. Ссылка в середине последовательности	#P1 = A + P2 + C; #P2 = B;	ABC	ABC (#P1), B (#P2)
29. Необязательная ссылка в середине последовательности	#P1 = A + ?P2 + C; #P2 = B;	AC	AC (#P1)
30. Повторяющаяся ссылка	#P1 = A + [2]P2 + C; #P2 = B;	ABBC	ABBC (#P1), B_1 (#P2), B_2 (#P2)
31. Цепочка ссылок	#P1 = A + P2; #P2 = B + P3; #P3 = C;	ABC	ABC (#P1), BC (#P2), C (#P3)
32. Несколько ссылок на один шаблон	#P1 = A + P3 + C; #P2 = A + P3; #P3 = B + B;	ABBC	ABBC (#P1), ABB (#P2), BB (#P3)
33. Вариация ссылок	#P1 = A + {P2, P3} + C; #P2 = B; #P3 = B + C;	ABCC	ABCC (#P1), B (#P2), BC (#P3)
34. Ссылка в исключении	#P1 = {A, ~P2}; #P2 = A + B;	AAB	A (#P1), AB (#P2)
35. Ссылка в вариации с исключением	#P1 = {P2, ~(A + B + C)}; #P2 = A + B;	ABABC	A_0B (#P1), A_0B (#P2), A_2B (#P2)
36. Правая рекурсия	#P = A + ?P	AAA	AAA

37. Левая рекурсия	#P = ?P + ?A	AAA	AAA
38. Шаблон внутри области действия	#P1 = B @ P2; #P2 = A + B + C;	ABCB	B (#P1), ABC (#P2)
39. Шаблон на границе области действия	#P1 = A @ P2; #P2 = A + B + C;	ABCB	A (#P1), ABC (#P2)
40. Шаблон, покрывающий всю область действия	#P1 = (A + B) @ P2; #P2 = A + B;	AB	AB (#P1), AB (#P2)
41. Необязательный шаблон в области действия	#P1 = A + ?B @ P2; #P2 = A + B + C;	ABCAB	AB (#P1), A_3 (#P2)
42. Вложенные области действия	#P1 = A @ P2 @ P3 @ P4; #P2 = B + A + B; #P3 = C + B + A + B + C; #P4 = D + C + B + A + B + C + D;	DCBABCD	A (#P1), BAB (#P2), CBABC (#P3), DCBABCD (#P4)

43. Ссылка в середине последовательности; один тег	#P1 = A + P2 + C;
	P2 = B;
		ABC	ABC (#P1)

44. Повторяющаяся ссылка; один тег	#P1 = A + [2]P2 + C;
	P2 = B;
		ABBC	ABBC (#P1)

45. Цепочка ссылок; один тег	#P1 = A + P2;
	P2 = B + P3;
	P3 = C;
		ABC	ABC (#P1)

46. Несколько ссылок на один шаблон; один тег	#P1 = A + P3 + C;
	P2 = A + P3;
	P3 = B + B;
		ABBC	ABBC (#P1)

47. Вариация ссылок; один тег	#P1 = A + {P2, P3} + C;
	P2 = B;
	P3 = B + C;
		ABCC	ABCC (#P1)

48. Ссылка в исключении; один тег	#P1 = {A, ~P2};
	P2 = A + B;
		AAB	A (#P1)

49. Ссылка в вариации с исключением; один тег	#P1 = {P2, ~(A + B + C)};
	P2 = A + B;
		ABABC	A_0B (#P1)

50. Шаблон внутри области действия; один тег	#P1 = B @ P2;
	P2 = A + B + C;
		ABCB	B (#P1)

51. Шаблон на границе области действия; один тег	#P1 = A @ P2;
	P2 = A + B + C;
		ABCB	A (#P1)

52. Шаблон, покрывающий всю область действия; один тег	#P1 = (A + B) @ P2;
	P2 = A + B;
		AB	AB (#P1)

53. Необязательный шаблон в области	#P1 = A + ?B @ P2;	ABCAB	AB (#P1)
действия; один тег	P2 = A + B + C;		

54. Вложенные области действия; один тег	#P1 = A @ P2 @ P3 @ P4;
	P2 = B + A + B;
	P3 = C + B + A + B + C;
	P4 = D + C + B + A + B + C + D;
		DCBABCD	A (#P1)

Испытания производительности.

Были выполнены испытания производительности созданного варианта осуществления изобретения. В испытаниях сравнивались времена поиска шаблонов и эквивалентных им регулярных выражений на одном и том же наборе текстов. Использовалось следующее окружение испытательного стенда:

ОС: Windows 10x64;

программная платформа: .NET Core 2.1.104;

среда выполнения программ: .NET Shared Framework Host 2.0.5;

процессор: Intel Core i7-8700 3.20 ГГц;

ОЗУ: 64 ГБ.

В качестве набора текстов использовались 19 текстовых файлов суммарным объемом 207 316 символов. Файлы были созданы из статей с новостных порталов, посвященных бизнесу и финансам. В тестах использовалась стандартная реализация регулярных выражений, входящая в поставку используемой платформы .NET Core.

В рамках испытания происходил замер непосредственно времени выполнения поиска. И текстовые данные, и шаблоны предварительно загружались в память. Это увеличивало точность сравнения, исключало длительные обращения к диску, время которых может значительно отличаться от теста к тесту. Создание объектов регулярных выражений и трансляция шаблонов также происходили на этапе подготовки к испытанию. Для уменьшения влияния динамической компиляции кода, использующейся в .NET, на

этапе подготовки также запускались несколько итераций поиска.

Сравнение производилось по трем различным классам шаблонов:

вариации лексем;

слова на расстоянии;

сложные шаблоны: телефон, URL-адрес, адрес электронной почты, хэш-тег.

Для первых двух классов шаблоны были сгенерированы автоматически на основе названий компаний и их биржевых меток (тикеров). Для каждого из тестов было сгенерировано 3383 шаблона. Далее в примерах символом A обозначается полное название компании, а символом B - её биржевой тикер.

Шаблоны вариаций лексем выявляют упоминание компаний. На языке описания шаблонов они имеют вид

```
{A, B}
```

Эквивалентные им регулярные выражение имеют вид

```
(?i)\bA\b|\bB\b
```

где директива (?i) активирует режим регистро-независимого поиска, a\b обозначает границу слова.

Шаблоны слов на расстоянии выявляют упоминания компании в контексте её биржевого тикера. Упоминанием считается нахождение в тексте названия компании и ее биржевого тикера на расстоянии не более пяти слов. На языке описания шаблонов выражение для поиска такой конструкций имеет вид

```
{A .. 0-5 .. B, B .. 0-5 .. A};
```

Схожее по смыслу регулярное выражение из-за громоздкости целиком не приводится. Для выражения конструкции следования на расстоянии не более пяти слов используется регулярное выражение

```
(\bA\b)(?:[\s,.:;!?()]+\w+){0,5}?[\s,.:;!?()]+(\bB\b)
```

где между A и B описывается конструкция из буквенных или буквенно-цифровых последовательностей, разделённых специальными или пробельными символами, повторяемая не более пяти раз.

Сложные шаблоны представляют собой написанные вручную шаблоны для выявления телефонного номера, URL-адреса, адреса электронной почты и хэш-тега. Для языка описания шаблонов используются следующие определения:

```
#PhoneNumber = ?"+" + {Num, "(" + Num + ")"} + [2+]({"-", Space} + Num);
#Email = Word + [0+] {Word, ".", "_", "+", "-"} + "@" + Domain;
#Url = {"http", "https"} + "://" + Domain + ?Path + ?Query;
Domain = Word + [1+] ("." + Word + [0+]{Word, "_", "-"});
Path = "/" + [0+] {Word, "/", "_", "+", "-", "%"};
Query = "?" + [0-1] (QueryParam + [0+]("&" + QueryParam));
QueryParam = Identifier + "=" + Identifier;
Identifier = {AlphaNum, Alpha, "_"} + [0+]{Word, "_"};
#HashTag = "#" + Identifier;
```

На языке регулярных выражений телефонный номер, адрес электронной почты, URL-адрес и хэш-тег описываются, соответственно, конструкциями

```
(?<=\s)\+?(\d+|\(\d+\))([-\s]\d+){2,}(?=\s)
[a-zA-Z0-9_.+-]+@([\w-]+(?:\.[\w-]+)*)
(https?:\/\/)([\w-]+(?:\.[\w-]+)*)(\/[\w\/%+-
]+)?(?:\?((\w+=\w+)(?:&(\w+=\ w+))*))?
\B(\#[a-zA-Z]+\b)
```

В ходе выполнения испытательных тестов и определения временных значений поиска для всех трех наборов шаблонов было скорректировано как количество испытательных запусков для вычисления среднего значения, так и количество холостых итераций. Время работы тестов поиска по шаблонам и теста поиска сложных конструкций с помощью регулярных выражений не превышает секунды, поэтому для увеличения точности использовалось десять холостых запусков и двадцать измеряемых итераций. Поиск с помощью регулярных выражений для тестов с большим количеством шаблонов выполняется более десяти секунд, поэтому запуск холостых: итераций не выполнялся, а для измерения использовались три итерации.

Результаты измерений и округлённое значение n, показывающее отношение времени поиска по регулярным выражениям к времени поиска по шаблонам, приведены в следующей таблице.

Тип испытательного теста	Время поиска по регулярным выражениям, секунд	Время поиска по шаблонам, с	*n*
Вариации констант	14	0,047	298
Следование через слова	15	0,2	75
Сложные конструкции	0,051	0,155	0,33

Результаты испытаний оказались предсказуемыми: в двух первых тестах при поиске 3383 конструкций поиск по шаблонам оказался значительно быстрее. Это связано как с возможностью выявления совпадений с множеством шаблонов за один просмотр текста, так и с работой на уровне лексем. В тесте поиска четырех сложных конструкций, где работа идет на уровне, близком к уровню символов, поиск по шаблонам оказался медленней всего лишь примерно в три раза.

Заключение.

Таким образом, обычному специалисту в данной области должно быть очевидно, что устройство, построенное в соответствии с предложенным изобретением, обеспечивает быстрый поиск в тексте совпадений с шаблонами за счёт предварительного разбора текста на лексемы и проверки всех совпадений всех шаблонов за один последовательный просмотр лексем текста. Достигаются высокая скорость, полнота и точность поиска, независимость поиска от языка текста и безопасность, означающая избежание затягивания поиска на продолжительное время, воспринимаемое пользователем как зацикливание или бесконечный поиск. Устройство, построенное в соответствии с предложенным изобретением, позволяет быстро выявлять наборы понятий, сущностей и их отношений в текстах на естественном языке.

Изобретение раскрыто в некоторых вариантах реализации. Должно быть очевидно, что раскрытое устройство может быть реализовано с модификациями, не отступая от изобретения. Приложенная формула изобретения покрывает всё множество вариаций и модификаций, находящихся в рамках сущности настоящего изобретения.

ФОРМУЛА ИЗОБРЕТЕНИЯ

1. Способ поиска в тексте совпадений с шаблонами для выявления заданного набора понятий, сущностей и отношений в тексте на естественном или машиночитаемом языке, характеризующийся тем, что

A) текст предварительно разбирают на лексемы, к которым относятся, по крайней мере, слова и разделители слов;

B) на языке описания шаблонов создают набор шаблонов, соответствующих заданным понятиям, сущностям и отношениям, в котором каждый шаблон является формальной грамматикой, состоящей, по крайней мере, из последовательностей, и(или) вариаций, и(или) повторений лексем текста, и(или) вхождений других шаблонов;

C) транслируют набор шаблонов в деревья поисковых выражений с поисковыми индексами, которые позволяют по заданной лексеме или заданному идентификатору шаблона быстро отыскать все поисковые выражения, которые начинаются с заданной лексемы или заданного шаблона;

D) для заданного набора шаблонов создают набор кандидатов, каждый из которых хранит информацию о сопоставлении лексем текста с элементами дерева поискового выражения на предмет совпадения, причём порядок сопоставления соответствует очерёдности обхода дерева от листьев к корню, совпадение последовательности требует совпадения всех её элементов в заданном порядке, совпадение вариации требует совпадения хотя бы одного её элемента, совпадение повторения требует совпадения его элемента заданное число раз, совпадение лексемы текста выполняется, по крайней мере, с учётом или без учёта заглавных и строчных букв;

E) затем единожды последовательно просматривают лексемы текста и для каждой лексемы выполняют, по крайней мере, следующие действия:

i) в поисковых индексах отыскивают все шаблоны, начинающиеся с текущей лексемы, создают кандидатов для проверки совпадений текста с этими шаблонами и добавляют их в набор кандидатов;

ii) для каждого кандидата из набора кандидатов текущую лексему текста сопоставляют с очередным элементом или элементами дерева поискового выражения кандидата на предмет совпадения;

iii) если очередной элемент дерева поискового выражения совпал с текущей лексемой текста и является последним в очерёдности обхода, например корневым, то кандидата считают полностью совпавшим и переносят из набора кандидатов в набор результатов; если очередной элемент дерева поискового выражения совпал с текущей лексемой текста и не является последним в очерёдности обхода, то кандидата считают частично совпавшим и оставляют в наборе кандидатов; если очередной элемент дерева поискового выражения не совпал с текущей лексемой текста, то кандидата считают несовпадающим и удаляют из набора кандидатов;

iv) для учёта различных вариантов совпадения текста с проверяемыми шаблонами создают и добавляют в набор кандидатов логические копии тех кандидатов, для которых возможны различные варианты совпадения с текстом, причём логические копии кандидатов содержат одинаковую информацию о совпадении элементов дерева поискового выражения с уже просмотренными лексемами текста и разнящуюся

информацию о совпадении элемента или элементов дерева поискового выражения с текущей лексемой текста.

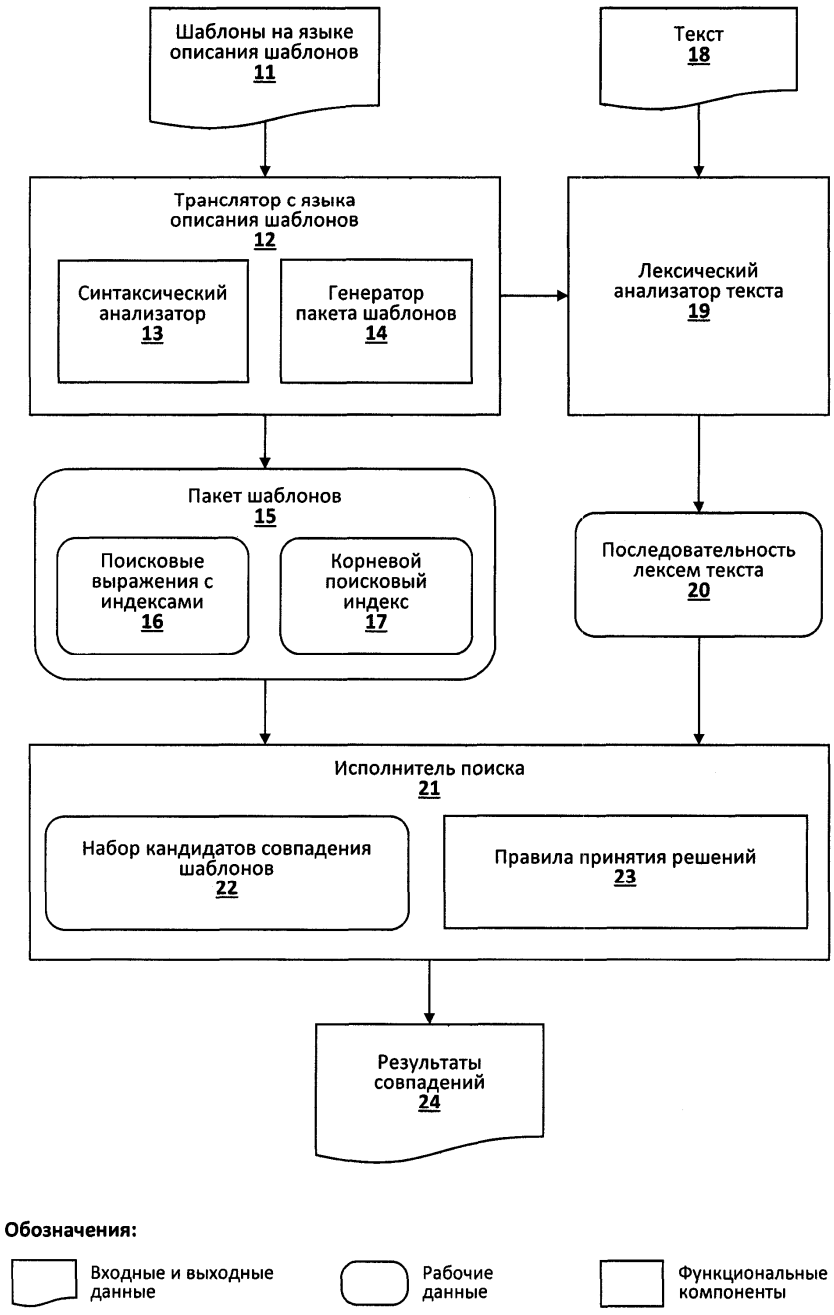
2. Способ поиска в тексте совпадений с шаблонами по п.1, в котором

A) проверяемые шаблоны допускают рекурсивные определения через те же самые шаблоны и(или) через другие шаблоны;

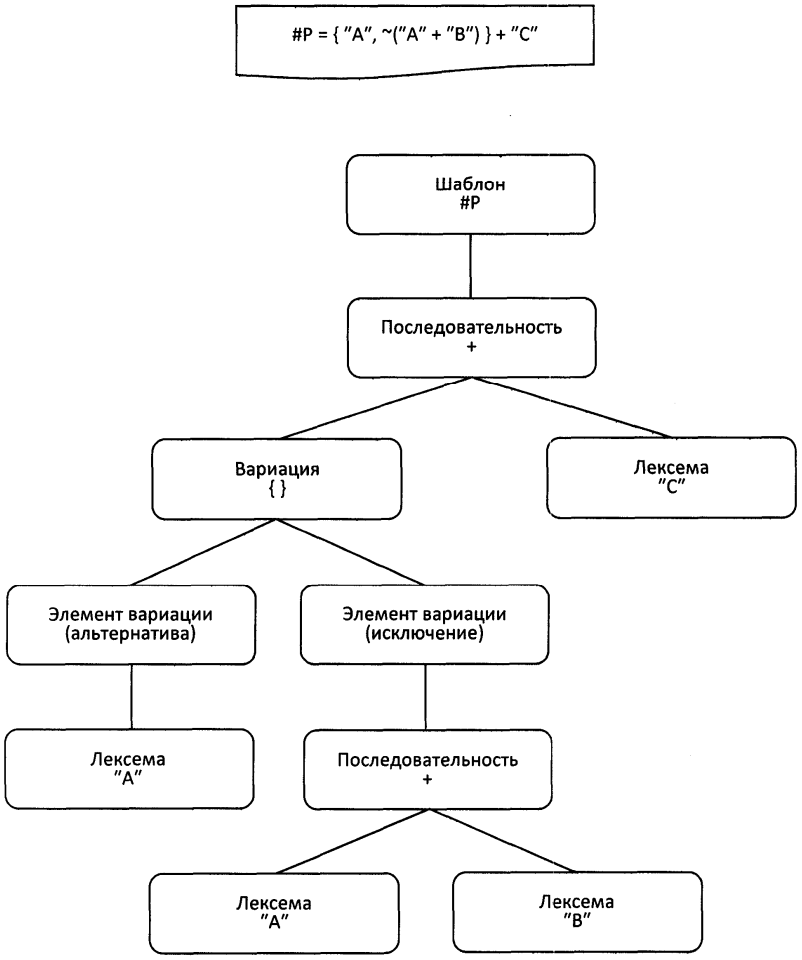
B) ограничено количество создаваемых в процессе поиска кандидатов и(или) объём потребляемых кандидатами ресурсов и(или) глубина рекурсии при переборе вариантов совпадений;

3. Способ поиска в тексте совпадений с шаблонами по п.1, в котором шаблоны поддерживают параметры, предназначенные для обобщения шаблонов и(или) для уточнения результатов поиска.

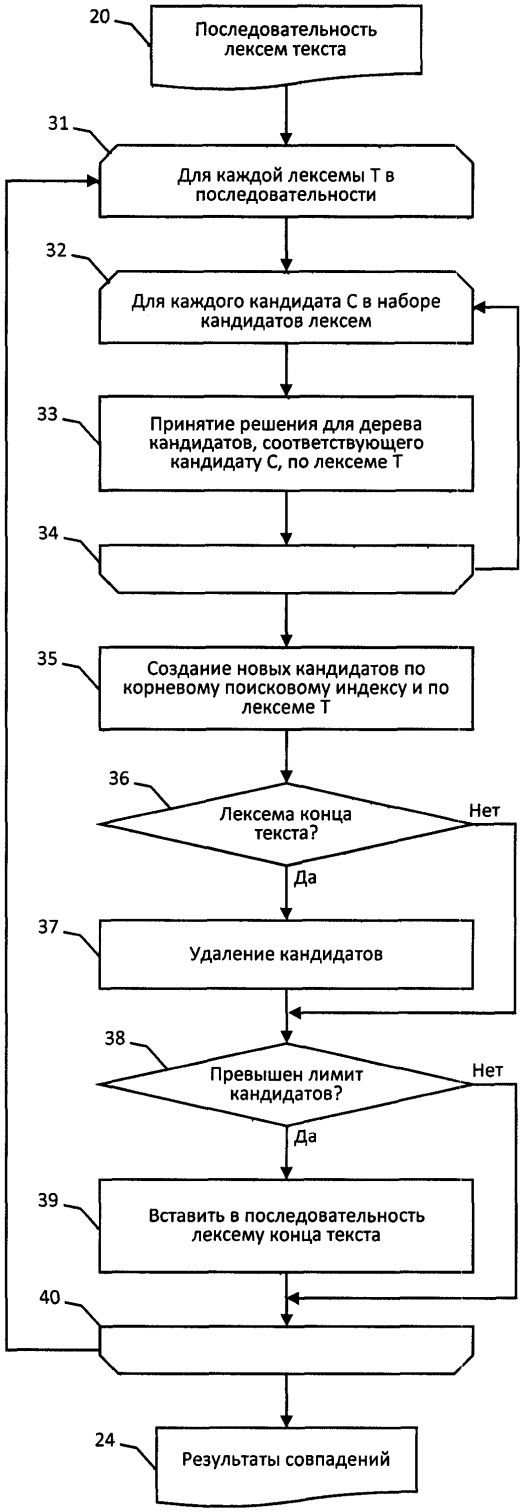
4. Способ поиска в тексте совпадений с шаблонами по п.2, в котором шаблоны поддерживают параметры, предназначенные для обобщения шаблонов и(или) для уточнения результатов поиска.



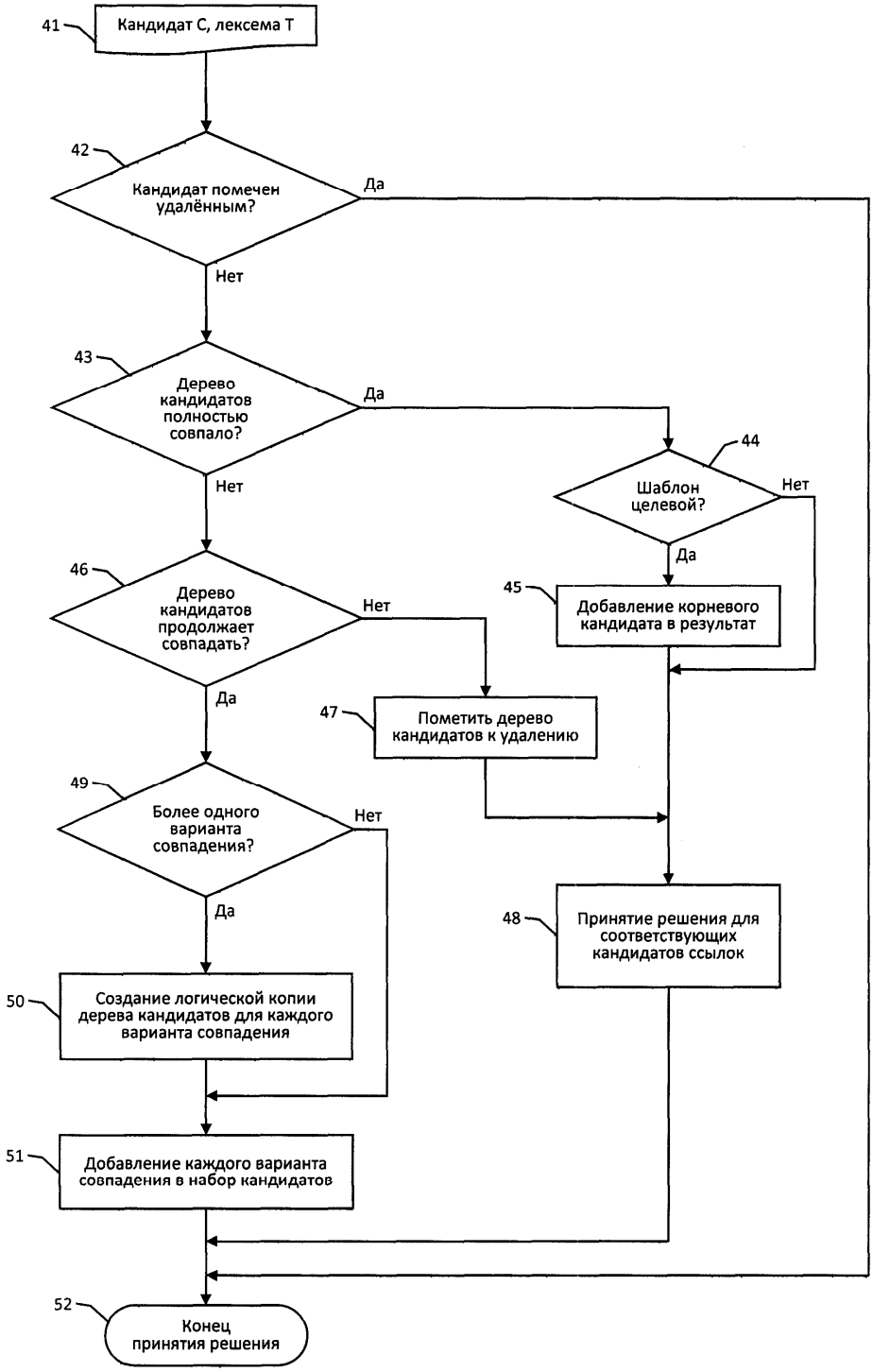
Фиг. 1


#P = { "A", ~("A" + "B") } + "C"
Шаблон
#P
Последовательность
+
Вариация
{ }
Лексема
"C"
Элемент вариации
(альтернатива)
Элемент вариации
(исключение)
Лексема
"A"
Последовательность
+
Лексема
"A"
Лексема
"B"

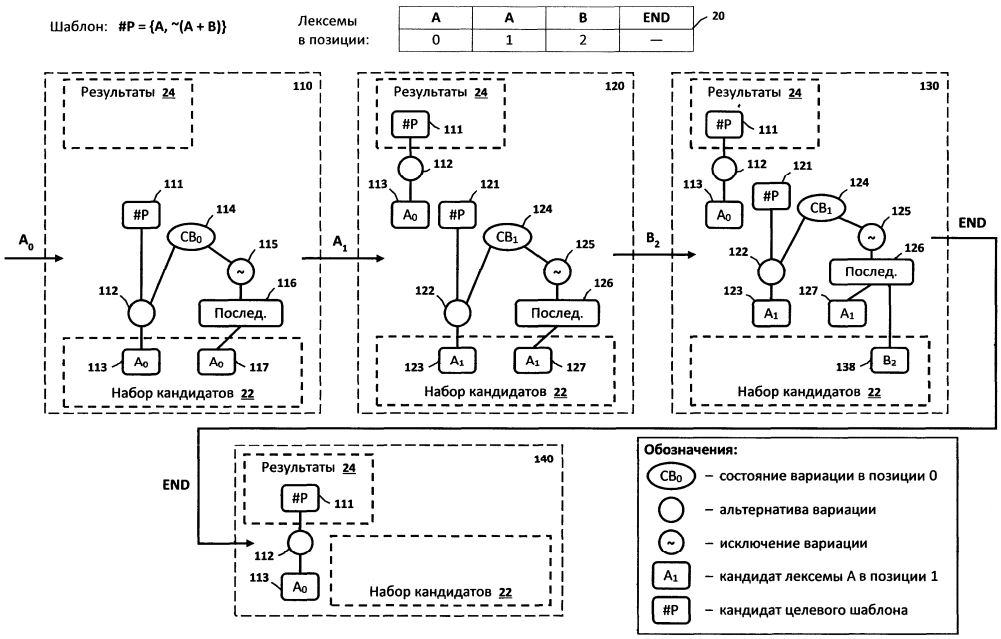
Фиг. 2


20
Последовательность лексем текста
31
Для каждой лексемы T в последовательности
32
Для каждого кандидата C в наборе кандидатов лексем
33
Принятие решения для дерева кандидатов, соответствующего кандидату C, по лексеме T
34
35
Создание новых кандидатов по корневому поисковому индексу и по лексеме T
36
Лексема конца текста?
Нет
Да
37
Удаление кандидатов
38
Превышен лимит кандидатов?
Нет
Да
39
Вставить в последовательность лексему конца текста
40
24
Результаты совпадений

Фиг. 3


41
Кандидат C, лексема T
42
Кандидат помечен удалённым?
Да
Нет
43
Дерево кандидатов полностью совпало?
Да
Нет
44
Шаблон целевой?
Нет
Да
45
Добавление корневого кандидата в результат
46
Дерево кандидатов продолжает совпадать?
Нет
Да
47
Пометить дерево кандидатов к удалению
49
Более одного варианта совпадения?
Нет
Да
48
Принятие решения для соответствующих кандидатов ссылок
50
Создание логической копии дерева кандидатов для каждого варианта совпадения
51
Добавление каждого варианта совпадения в набор кандидатов
52
Конец принятия решения

Фиг. 4


Шаблон: #P = {A, ~(A + B)}
Лексемы в позиции:
A | A | B | END
0 | 1 | 2 | —
20
Результаты 24
110
120
130
140
Набор кандидатов 22
#P
CB0
CB1
Послед.
A0
A1
B2
END
Обозначения:
– состояние вариации в позиции 0
– альтернатива вариации
– исключение вариации
– кандидат лексемы A в позиции 1
– кандидат целевого шаблона

Фиг. 5