Оптимизация вычисления обратных осей языка XML Path при его реализации функциональными методами

Д.А. Лизоркин
(МГУ, ИСП РАН, lizorkin@ispras.ru)

Аннотация. XPath – это язык для адресации структурных частей XML-документа. Функциональный язык программирования Scheme позволяет естественным образом представлять и обрабатывать XML-документы в виде SXML и обеспечивает единую среду для реализации XML-приложений.

Ограничением SXML – абстрактного синтаксического дерева XML-документа в форме S-выражения – является отсутствие указателей с дочерних узлов на родительские узлы, что затрудняет вычисление обратных осей языка XPath над SXML-документом. В работе предлагается алгоритм, позволяющий построить вычисление выражений XPath таким образом, что наличие указателей с дочерних узлов на родительские узлы в дереве документа становится необязательным. Проводится обоснование алгоритма и рассматриваются его основные свойства. Предлагаемый в работе подход оптимизирует вычисление обратных осей языка XPath над SXML-документами, что подтверждается результатами проведенных экспериментов.

1. Введение

Язык XML Path (XPath) [1], разработанный Консорциумом Всемирной Сети, – это язык для адресации структурных частей XML-документа. Язык XPath является ключевым языком для платформы XML и изначально разрабатывался как основа для нескольких других языков обработки XML-данных; в частности, XSLT, XPointer и XQuery.

Поскольку большинство языков платформы XML не являются языками программирования общего назначения, при реализации законченных XML-приложений они обычно используются совместно с некоторым традиционным языком программирования. Комбинирование двух различных по своей природе языков (например, XPath и Java) приводит к проблеме, известной как несоответствие импеданса (impedance mismatch) [2].

Барьер между языками платформы XML и языками программирования общего назначения может быть снят за счет обработки XML-данных функциональными методами. В основе данного подхода лежит SXML – реализация Информационного Пространства XML в виде S-выражений [3].

Язык функционального программирования Scheme семейства Лисп использует S-выражения для представления и своих данных, и своего кода, что позволяет создать единую среду для написания XML-приложений. Язык Scheme [4] – это один из самых лаконичных и компактных языков, применяемых на практике, и получил широкое признание как скрипт-язык [5]. На Scheme были реализованы инструменты для обработки SXML-документов, совместимые со спецификациями Консорциума Всемирной Сети и обеспечивающие бесшовную интеграцию языков платформы XML с языком программирования высокого уровня.

Язык XPath предоставляет приложению возможность навигации по иерархической структуре XML-документа [1]. В частности, приложению, производящему обработку XML-данных, может потребоваться выбрать родительский узел или более далеких предков для данного узла дерева XML-документа. Для обеспечения заданной функциональности спецификация языка XPath предоставляет обратные оси – такие как parent, ancestor и ряд других, – которые позволяют выбрать узлы, предшествующие данному узлу в порядке обхода узлов в дереве документа. Узел в Модели данных языка XPath [1] (и, в более общем случае, информационная единица в Информационном Пространстве XML [6]) имеет свойство “родитель”, представляющее собой указатель с данного узла на его родительский узел. Документ в SXML является S-выражением и поэтому не обладает подобными указателями, поскольку S-выражения моделируют ориентированные деревья, по определению не имеющие указателей в направлении к корню [7].

Может казаться, что с помощью SXML нельзя полностью смоделировать Информационное Пространство XML и Модель данных XPath. Так, в SXPath – реализации функциональными методами языка XPath для документов на SXML – до настоящего времени проводилось очень неэффективное во временном отношении вычисление обратных осей XPath, что обусловлено отсутствием указателей с дочерних узлов на родительские узлы в SXML. Однако в данной статье мы покажем, что возможно организовать вычисление выражений XPath таким образом, что наличие указателей с дочерних узлов на родительские узлы становится необязательным. В работе предлагается алгоритм, оптимизирующий вычисление обратных осей языка XPath над SXML-документами и документами, не имеющими указателей с дочерних узлов на родительские узлы. Предложенный алгоритм был полностью реализован на языке функционального программирования Scheme как расширение к SXPath. Проведенные эксперименты свидетельствуют о значительном увеличении производительности SXPath при вычислении над SXML-документами выражений языка XPath, содержащих обратные оси.

Применение предлагаемого в статье алгоритма не ограничивается случаем, когда обрабатываемые древовидные данные представлены в виде SXML, но может использоваться и в других случаях – когда возвращение к родительскому элементу является невозможным или нежелательным. Например, предлагаемый алгоритм может оптимизировать вычисление выра-

жений языка XPath над потоковыми данными, поскольку обеспечивает возможность последовательного просмотра документа в одном направлении, без необходимости возвращаться к родительским элементам, являющимся с точки зрения порядка документа предшественниками для своих дочерних узлов.

Необходимо отметить, что, хотя рассуждения в данной статье проводятся для Рекомендации XPath версии 1.0, все полученные результаты полностью переносятся и на язык XPath версии 2.0 [8], черновая спецификация которого в настоящий момент проходит процесс перехода в статус Рекомендации консорциума Всемирной Сети.

Статья организована следующим образом. В разделе 2 дается обзор основных понятий, используемых в ходе дальнейшего изложения. Раздел 3 посвящен рассмотрению связанных работ по данной предметной области. В разделе 4 на простом примере иллюстрируется основная идея предлагаемого в данной работе алгоритма. Описание основного алгоритма и его обоснование даны в разделе 5. В разделе 6 обсуждаются свойства предложенного алгоритма; ограничения алгоритма рассматриваются в разделе 7. Результаты проведенных экспериментов обсуждаются в разделе 8. Раздел 9 завершает статью.

2. Обзор предметной области

Данный раздел дает обзор основных понятий, которые будут использоваться в ходе последующего изложения. Основное внимание в данной статье сосредоточено на языке XPath; обзор SXML и SXPath приведен для иллюстрации инструментария, для которого был разработан предлагаемый в данной работе алгоритм оптимизации вычисления обратных осей XPath.

2.1. Обзор XPath

Назначение языка XPath – адресация структурных частей XML-документа. Ввиду того, что XML-документ является, в сущности, древовидной структурой, модель данных языка XPath [1] представляет документ как дерево узлов.

Главной синтаксической конструкцией языка является выражение (expr), которое соответствует одноименному правилу грамматики. Вычисление выражения осуществляется относительно контекста. Контекст включает в себя:

- Узел (который мы далее будем называть контекстным узлом);
- Набор связанных переменных;
- Библиотеку базовых функций XPath [9];
- А также несколько других составляющих, которые в данной статье мы затрагивать не будем.

Несмотря на то, что выражение (expr) является самой общей конструкцией языка XPath, самой важной конструкцией языка считается путь доступа (location path) [1]. Путь доступа применяется к контекстному узлу, и результатом вычисления является набор узлов (node-set) [9], состоящий из (возможно, нескольких) узлов, выбранных с помощью данного пути доступа

относительно контекстного узла. Выбранные узлы соответствуют элементам, атрибутам, текстовым данным и другим частям XML-документа.

Путь доступа состоит из последовательности одного или более шагов доступа (location step), синтаксически отделяемых друг от друга символом косой черты ("/"). Шаг доступа включает в себя 3 составляющие:

- **Ось** (axis), определяющую соотношение в дереве между узлами, в контексте которых вычисляется шаг доступа, и узлами, которые выбирает шаг доступа. Ось можно считать "направлением движения" по дереву, представляющему XML-документ [10]. Спецификация XPath определяет 13 различных осей. Они включают в себя оси для спуска к листьям дерева, для подъема в сторону корня, для выбора соседних узлов и т.п. Синтаксически имя оси отделяется от остальной части шага адресации с помощью двойного двоеточия ("`::`").
- **Тест узла** (node test), который определяет тип и, возможно, имя узлов, выбираемых шагом доступа. В то время как ось определяет "направление движения", тест узла определяет желаемые узлы, которые должны быть выбраны.
- Ноль или более **предикатов** (predicates). Каждый предикат синтаксически записывается в квадратных скобках и используется для дальнейшего просеивания набора узлов, выбираемых шагом доступа.

Шаги в пути доступа вычисляются по очереди слева направо. Самый левый шаг вычисляется первым, обычно по отношению к узлу, который представляет корень XML-документа. Каждый последующий шаг доступа выбирает набор узлов, который вычисляется по отношению к набору узлов, выбранному предыдущим шагом доступа. Набор узлов, выбранный самым правым шагом доступа – это результат всего пути доступа для данного XML-документа.

Пример пути доступа языка XPath приведен на рис. 1. Данный путь доступа состоит из 3 шагов доступа, и в последних 2 шагах имеется по одному предикату. Ввиду описанной выше семантики вычисления шагов в пути доступа, легко видеть, что путь доступа на рис. 1 выбирает 2-й раздел (section) 5-й главы (chapter) элемента документа с именем doc.

```
/child::doc/child::chapter[position()=5]/child::section[position()=2]
```

Рис. 1. Пример пути доступа, выбирающего 2-й раздел (`section`) 5-й главы (`chapter`) элемента документа `doc`.

2.2. Обзор SXML

Языки SXML и XML могут рассматриваться как два синтаксически различных представления Информационного Пространства XML Infoset [6].

Язык XML использует язык разметки SGML для представления информационных единиц Информационного Пространства XML и их свойств.

Древовидная структура документа (свойства “родитель” и “ребенок” информационных единиц Информационного Пространства XML) выражается при помощи вложенных тегов разметки [5].

Язык SXML использует для представления информационных единиц Информационного Пространства XML и их свойств S-выражения языка Scheme. Древовидная структура документа выражается при помощи вложенных списков. Каждая из информационных единиц Информационного Пространства XML представляется в виде S-выражения, первым элементом которого является либо имя информационной единицы (для типов “элемент” и “атрибут”), либо служебное имя, предусмотренное для информационной единицы данного типа в грамматике SXML [11].

Пример простого XML-документа и его представления на SXML приведены на рис. 2, наглядно демонстрирующем соответствие между вложенными тегами XML и вложенными списками SXML. Более подробно ознакомиться с SXML можно в [2].

```
<?xml version='1.0'?>
<doc>
  <tag attr1="value1" attr2="value2">
    <nested>Text node</nested>
  </tag>
  <empty/>
</doc>
```

```
(*TOP* (*PI* xml "version='1.0'")
 (doc
   (tag (@ (attr1 "value1") (attr2 "value2"))
     (nested "Text node")
   )
                 (empty)

 ))
```

Рис. 2. XML-документ (левый столбец) и его представление в SXML.

2.3. Обзор SXPath

SXPath – это реализация XPath на языке функционального программирования Scheme, предоставляющая язык запросов к документам на SXML. Реализация SXPath трактует путь доступа как составной запрос к дереву документа или его ветви. Отдельный шаг доступа представляет собой комбинацию проекции, выборки или транзитивного замыкания [*12*]. Несколько шагов доступа комбинируются с помощью операций последовательного применения или объединения.

Библиотека SXPath состоит из набора низкоуровневых предикатов, фильтров, операций выборки и комбинаторов; и функций высокого уровня, реализованных в терминах низкоуровневых функций. На высоком уровне предоставляется возможность записи запросов в двух различных нотациях:

- Запрос может быть записан в виде списка, состоящего из шагов доступа. В качестве шага доступа может использоваться произвольный преобразователь – в том числе и произвольная пользовательская функция с заданной сигнатурой. Список шагов доступа транслируется в комбинацию примитивов SXPath с помощью набора правил перезаписи.

- Запрос может представлять собой выражение XPath, записанное в виде текстового синтаксиса, полностью совместимого со Спецификацией XPath Консорциума Всемирной Сети.

Для каждой из описанных выше нотаций в библиотеке SXPath существует собственная функция высокого уровня, конструирующая по запросу в данной нотации его реализацию на языке программирования Scheme. Конструируемая реализация запроса затем может быть применена к SXML-документу и вычисляет данный запрос над данным документом. В силу описанного дизайна SXPath обе высокоуровневые функции, конструирующие по запросу его реализацию, могут рассматриваться в качестве компиляторов из пользовательской нотации запросов в комбинацию примитивов, в роли которых служат низкоуровневые функции SXPath [*12*].

Следует заметить, что обе нотации могут комбинироваться внутри одного запроса, что позволяет комбинировать синтаксис, совместимый со Спецификацией XPath Консорциума Всемирной Сети, и возможности языка программирования общего назначения Scheme [3].

3. Родственные работы по предметной области

При исследовании работ по данной предметной области нельзя не упомянуть статью [7], в которой производится конструктивное доказательство того, что SXML является полной моделью Информационного Пространства XML [6]. В статье предлагаются методы восстановления изоморфизма между SXML и моделью данных XPath.

В [7] утверждается, что, поскольку выражение XPath может включать абсолютный путь доступа, контекст вычисляемого выражения должен содержать корневой узел документа. На основании данного наблюдения делается вывод, что указатель с дочернего узла на родительский узел всегда (концептуально) присутствует в SXML. С целью нахождения родителя для данного узла предлагается производить поиск по всему дереву SXML вниз от корня, чтобы найти тот узел, одним из дочерних узлов которого является данный. В виде формулы предложенный метод нахождения родителя для узла x может быть записан следующим образом:

$$\text{parent}(x) = \{\, y \mid y=\text{child}^*(\text{root}),\ x=\text{child}(y) \,\}\ ,$$

где символ `child*` обозначает транзитивное замыкание оси `child`, а символом `root` обозначен корень дерева SXML-документа.

Очевидным недостатком поиска родительского узла от корня документа является его временная дороговизна, поскольку данный метод в общем случае требует сканирования всего дерева документа. Необходимо отметить, что до настоящего времени обратные оси в SXPath были реализованы именно с помощью метода поиска родительского узла от корня документа. Одной из

целей, достигнутых в настоящей работе, было повышение производительности SXPath за счет оптимизации вычисления обратных осей.

В статье [7] также предлагается 3 метода восстановления указателей к родительским узлам за счет присоединения аннотаций к дочерним узлам дерева SXML-документа. Каждый из данных методов имеет свои преимущества и недостатки для конкретной решаемой задачи, однако общим недостатком всех этих 3-х методов является значительное усложнение структуры данных, используемой для представления XML-документа в виде S-выражения. С добавленными к узлам аннотациями документ на SXML теряет свое важное преимущество – являться одновременно внутренним представлением данных и наглядной внешней нотацией. Указатели на родительские узлы, добавленные к документу в виде аннотаций, также усложняют обработку SXML-документа, поскольку превращают дерево документа в ориентированный граф с двунаправленными указателями. Как отмечается в [7], обработка циклических структур в чисто функциональном стиле программирования является далеко не простой задачей.

В настоящей работе предлагается алгоритм вычисления выражений XPath, не требующий изменения структуры данных, используемой для представления документа на SXML. Документ на SXML сохраняет свою простую и естественную структуру, а все действия, связанные с нахождением родительских узлов и вычислением обратных осей языка XPath, полностью инкапсулированы от пользователя внутри реализации предлагаемого алгоритма.

В статье [*13*] предлагается алгоритм, основанный на правилах перезаписи, позволяющий преобразовать путь доступа XPath в эквивалентный путь доступа, не содержащий обратных осей. Хотя в [*13*] решалась задача обеспечения потокового вычисления путей доступа XPath, полученные результаты в определенной степени могут быть использованы и в SXML для решения проблемы указателей на родительские узлы. Необходимо отметить, что правила перезаписи требуют предварительного расширения языка XPath дополнительным оператором сравнения узлов, которого нет в Спецификации XPath версии 1.0.

Алгоритм перезаписи, предложенный в [*13*], обладает тем недостатком, что не любое выражение XPath может быть с его помощью преобразовано в эквивалентное выражение, не содержащее обратных осей [7]. Предлагаемый в настоящей работе алгоритм вычисления обратных осей применим для вычисления произвольного выражения языка XPath без необходимости иметь указатели на родительские узлы в дереве документа.

4. Иллюстрация предлагаемого подхода

В данном разделе рассматривается пример, иллюстрирующий последующее изложение предлагаемого в данной статье алгоритма вычисления обратных осей языка XPath.

Пример 1. Рассмотрим следующее выражение языка XPath:

```
/doc/head/../body
```

Данное выражение представляет собой путь доступа (location path) и состоит из 4 шагов доступа:

- На первом шаге выбирается элемент документа, имеющий имя *doc*.
- На 2-м шаге выбирается дочерний элемент с именем *head*.
- На 3-м шаге возвращаемся к родительскому элементу элемента *head*.
- На последнем шаге переходим к дочернему элементу с именем *body*[1].

Заметим, что на 3-м шаге рассматриваемого нами пути доступа имеется обратная ось `parent` [8] (записанная в терминах сокращенного синтаксиса XPath в виде двух точек). Ввиду того, что в SXML отсутствуют указатели с дочерних узлов на родительские узлы, реализовать стратегию прямолинейного пошагового вычисления данного пути доступа в SXML невозможно, поскольку, имея на входе лишь контекстный узел, невозможно получить никаких сведений об его родительском узле.

Даже не имея указателей с дочерних узлов на родительские узлы, тем не менее возможно вычислить путь доступа из примера 1, если задействовать дополнительные соображения о структуре вычисляемого выражения:

- На первом шаге доступа мы, как и прежде, выбираем элемент документа, имеющий имя `doc`.
- Допустим, что на 2-м шаге, когда нужно выбрать дочерний элемент с именем `head`, мы уже каким-то образом знаем, что следующим 3-м шагом потребуется возвращаться по дереву документа обратно, на тот самый узел, который на текущем 2-м шаге является контекстным узлом. Ввиду данного наблюдения, мы поступим более расчетливо: на 2-м шаге доступа не только выберем требуемый дочерний элемент с именем `head`, но также сохраним текущий контекстный узел, поскольку он потребуется при вычислении оси `parent` на следующем шаге.
- Благодаря проделанной предварительной подготовке, вычисление оси `parent` на 3-м шаге доступа теперь сводится к простому извлечению ранее сохраненного узла.
- Последний шаг вычисляется как обычно: мы выбираем дочерний элемент с именем `body`.

Благодаря анализу вычисляемого пути доступа из примера 1, содержащего обратную ось, оказалось возможным вычислить его даже при отсутствии указателей с дочерних узлов на родительские узлы в дереве документа. В следующем разделе рассмотренный на данном примере способ вычисления

[1] Данный пример служит исключительно для иллюстрации последующего изложения. Рассматриваемый путь доступа на XPath не несет семантической нагрузки, но сделан ради наглядности максимально простым.

обратных осей формализуется более строго и обобщается в виде алгоритма на случай произвольного выражения языка XPath.

5. Алгоритм вычисления выражений XPath, содержащих обратные оси

Идея алгоритма состоит в том, что по ходу вычисления выражения языка XPath мы будем запоминать те узлы обрабатываемого документа, которые являются предками для контекстного узла и которые потребуются при дальнейшем вычислении выражения. Как и сам контекстный узел, его предки будут храниться в контексте, над которым производится вычисление выражения XPath. В тот момент, когда вычисление выражения потребует адресации к родительскому узлу контекстного узла или другому его предку, требуемый узел будет сразу извлекаться из контекста. При данном подходе переход к предку контекстного узла не требует нового обращения к дереву документа благодаря тому, что требуемый узел уже был сохранен заранее. Мы можем корректно говорить о том, что при вычислении выражения XPath предки всегда посещаются раньше своих потомков, на основании следующего утверждения.

Утверждение 1. При вычислении выражения языка XPath над некоторым XML-документом, для того, чтобы выбрать в документе некоторый контекстный узел, всегда необходимо сначала посетить все узлы, которые являются для него предками.

Доказательство 1. Доказательство утверждения проведем индукцией по глубине документа. Поскольку вычисление выражения языка XPath начинается от корня документа, то базис индукции доказан, т.к. корень документа является предком для всех остальных узлов в документе.

В соответствии с Информационным Пространством XML Infoset [6], узлы XML-документа описываются информационными элементами, которые соединены между собой исключительно дочерними и родительскими связями. Поскольку вычисление выражения начинается от корневого узла и в дереве существуют только вертикальные связи между узлами, любой путь, позволяющий добраться до конкретного контекстного узла, с необходимостью будет проходить через его родительский узел; что доказывает индуктивный переход. Утверждение доказано.

Замечание 1. Некоторые реализации Информационного Пространства XML, например, Объектная Модель Документа (Document Object Model, DOM), дополнительно используют указатели и между соседними узлами-братьями, т.е. такими, которые имеют общий родительский узел. Наличие подобных горизонтальных связей не влияет на справедливость данного утверждения, поскольку при перемещении по горизонтальным связям родительский узел для контекстного узла остается постоянным. Данное замечание означает, что все последующие результаты в полной мере могут быть применены к Объектной Модели Документа и любой изоморфной ей модели.

Организация вычисления выражений языка XPath с использованием сохраняемых в контексте предков контекстного узла будет основана на специальной реализации осей языка XPath. Реализация обратных осей (таких как `parent`, `ancestor`, `preceding` и др.) будет обеспечиваться благодаря наличию в контексте предварительно сохраненных там узлов – предков контекстного узла. Для того, чтобы необходимые для обратных осей предки контекстного узла были сохранены в контексте, нам также потребуется специальная реализация и тех осей, которые осуществляют спуск по дереву документа[2]. Оси, осуществляющие спуск по дереву документа (например, оси `child`, `descendant`), должны отвечать также за то, чтобы сохранять в контексте узлы, которые были пройдены в процессе спуска (и которые станут предками для контекстного узла на последующих шагах доступа). Доказанное выше утверждение гарантирует, что предки контекстного узла всегда могут быть сохранены в контексте, до того как потребуется их использование.

При оптимизации вычисления обратных осей XPath за счет хранения в контексте предков контекстного узла желательно также добиться минимальности хранимой в контексте дополнительной информации. Мы будем стремиться к тому, чтобы хранить в контексте только тех предков контекстного узла, которые действительно потребуются при дальнейшем вычислении выражения XPath, и не включать в контекст остальных предков. Утверждается, что желаемого результата можно добиться за счет статического анализа выражения XPath. Под **фазой статического анализа** понимается такая фаза обработки, на которой выражение XPath анализируется без учета входных данных; в отличие от следующей за ней **фазы динамического вычисления**, когда выражение вычисляется по отношению к конкретному XML-документу.

5.1. Количество предков для контекстного узла

Предки контекстного узла, которых мы будем сохранять в контексте, будут определяться с помощью целого неотрицательного числа, которое назовем **количеством предков.**

Определение 1. Количество предков – это целое неотрицательное число или $+\infty$:

ancestors_number = 0, 1, 2, ..., $+\infty$.

В соответствии со значением этого числа, в контексте вычисляемого выражения языка XPath будут дополнительно храниться узлы {**node1, node2, ..., noden**}, где

- **node1** – родительский узел для контекстного узла;
- **nodek+1** – родительский узел для узла nodek, k=1, n-1;
- либо **n<ancestors_number** и **noden** – корневой узел дерева XML-документа, либо **n=ancestors_number**.

[2] Спуском по дереву документа считаем движение по направлению от корня документа к его листовым узлам.

Из определения следует, что при количестве предков равном $+\infty$ в контексте должны сохраняться все предки контекстного узла до корневого узла включительно.

В предлагаемом алгоритме мы будем определять количество предков, которое требуется для вычисления данного подвыражения XPath (в его взаимоотношении с другими подвыражениями анализируемого выражения). В соответствии с данным выше определением, количество предков будет однозначно задавать тех предков контекстного узла, которых необходимо сохранить в контексте для корректного вычисления данного подвыражения[3].

В качестве единицы подвыражения XPath мы возьмем грамматическое правило языка XPath. Большинство грамматических правил может включать в своем определении другие правила, например, путь доступа (location path) включает в себя несколько шагов доступа (location step). Можно говорить о том, что предлагаемый алгоритм рассматривает выражение языка XPath в виде абстрактного синтаксического дерева: вершинами этого дерева служат грамматические правила языка XPath, а дугами – отношение включения между правилами. Если рассматривать выражение XPath в виде подобного абстрактного синтаксического дерева, то идея алгоритма заключается в обходе данного дерева и в приписывании каждой его вершине такого количества предков, которое необходимо для вычисления соответствующего подвыражения XPath.

Для уменьшения объема выкладок будем предполагать, что все конструкции сок-ращенного синтаксиса XPath были расширены до полного синтаксиса. Данное предположение не ограничивает общности предлагаемых рассуждений, пос-кольку разумно проводить расширение сокращенного синтаксиса XPath на этапе разбора выражения, т.е. уже в процессе построения его синтаксического дерева.

Считаем, что результатом фазы статического анализа выражения XPath является генерация некоторого кода, который в соответствии с заданным выражением на последующей фазе динамического выполнения вычислит требуемый результат над некоторым входным XML-документом. Поскольку выражение языка XPath в соответствии с грамматикой XPath может быть представлено в виде абстрактного синтаксического дерева, можно говорить о том, что каждому поддереву в этом дереве также соответствует некоторый кусок кода, который генерируется из этого поддерева.

Определение 2. Будем говорить, что от некоторой вершины абстрактного синтаксического дерева выражения XPath требуется количество предков, равное N, понимая под этим то, что кусок кода, который будет сгенерирован для поддерева, соответствующего данной вершине, обязан после своей работы сохранить N предков в возвращаемом контексте.

[3] Корректным вычислением выражения XPath будем называть такое вычисление, который всегда приводит к результату, соответствующему Спецификации языка XPath.

Определение 3. Будем говорить, что вершина абстрактного синтаксического дерева требует для себя количество предков, равное M, понимая под этим то, что куску кода, который будет сгенерирован для поддерева, соответствующего данной вершине, для корректной работы необходимо предоставить во входном контексте M предков. Корректная работа включает в себя также необходимость сохранения в возвращаемом контексте N предков, которые от данной вершины требуются.

Для компактности записи правил алгоритма будем использовать нотацию функции, имеющей следующую сигнатуру:

XPath_grammar_rule(required_ancestors_number) = requires_ancestors_number ,

где

- в качестве имени функции будет использоваться имя грамматического правила XPath, которое стоит в данной вершине абстрактного синтаксического дерева;
- аргументом функции является количество предков, которое требуется от данной вершины;
- возвращаемым результатом функции является количество предков, которое данная вершина требует для себя (и которое естественным образом зависит от аргумента функции в соответствии с определением 3).

Необходимо заметить, что поскольку тип аргумента и возвращаемого результата функции совпадают, правомерно рассматривать суперпозиционную комбинацию функций рассмотренной сигнатуры, что семантически будет соответствовать отношению включения грамматических правил языка XPath друг в друга.

Пример 2. Пусть в некоторой вершине абстрактного синтаксического дерева выражения языка XPath стоит спецификатор оси *child*, и от данной вершины требуется сохранение 2 предков (т.е. родителя и прародителя контекстного узла). Поскольку ось *child* выбирает дочерние узлы для контекстного узла, то контекстный узел является родителем для узлов, которые будут получены в результате применения оси *child* к контекстному узлу. Ввиду данного наблюдения, реализация оси *child* может по требованию сохранить в контексте результата:

- Входной контекстный узел, который стал родителем для результирующего контекстного узла;
- Если во входном контексте были сохранены узлы-предки контекстного узла, то они также могут быть сохранены в результирующем контексте, соответственно как прародитель, прапрародитель и т.д.

Легко видеть, что реализация оси *child*, имея на входе контекст, содержащий лишь контекстный узел (и не хранящий никаких его узлов-предков), в качестве результата возвращает контекст, в котором по требованию может быть сохранен родитель результирующего контекстного узла. Если же требуется сохранить большее количество предков, то тогда вершина абстрактного

синтаксического дерева, в которой стоит спецификатор оси *child*, должна потребовать для себя количество предков, на 1 меньшее, чем то количество предков, которые потребовали от нее. В принятых выше обозначениях, это может быть кратко записано в виде:

$$Child(ancestors\_number) = ancestors\_number - 1,$$

а для нашего примера: $Child(2) = 1$.

В терминах введенных выше определений, построение алгоритма вычисления выражений XPath на основе сохраняемых в контексте предков контекстного узла заключается в определении каждой функции, соответствующей каждому из правил грамматики XPath. Поскольку грамматические правила включают в себя **спецификатор оси** (axis specifier), рассуждения попутно будут включать в себя определение необходимого количества предков для вычисления каждой обратной оси языка XPath.

В том случае, когда количество предков принимает значение +∞, для него будут применяться стандартные математические соглашения о работе с бесконечностями:

$$+\infty - C = +\infty, \quad C \neq \infty;$$
$$\max(+\infty, C) = +\infty.$$

5.2. Описание алгоритма

Для анализируемого исходного выражения считаем, что для него требуется количество предков, равное 0, и анализируем подвыражения, из которого оно состоит, в соответствии с грамматикой языка XPath.

1. Для **пути доступа** (location path) будем рассматривать входящие в его состав шаги доступа в обратном порядке.

   $$LocationPath ::= Step_1 / Step_2 /.../ Step_{n-1} / Step_n$$

   Для последнего шага будем требовать, чтобы он сохранил в контексте такое количество предков, которое требуется от всего пути доступа. Для предпоследнего шага будем требовать сохранения такого количества предков, которое потребовал для себя последний шаг доступа. И так далее, пока мы не дойдем до первого шага доступа, и количество предков, которое он для себя потребует, будет тем количеством, которое требуется при вычислении всего пути доступа. Данные рассуждения можно компактно записать в виде суперпозиционной формулы:

   $$LocationPath(ancestors\_number) = Step_1(Step_2 (... (Step_{n-1} (Step_n(ancestors\_number))) ...)) \quad .$$

   Написанная формула в формальном виде отражает то наблюдение, что каждый шаг доступа должен сохранять в контексте определенное количество предков, руководствуясь теми шагами доступа, которые являются следующими после него в пути доступа.

2. В **шаге доступа** (location step) нас будут интересовать его спецификатор оси и предикаты. Тест узла (node test), входящий в состав шага доступа, нас интересовать не будет, поскольку для его вычисления не требуются знания о предках контекстного узла.

   $$Step ::= AxisSpecifier\ NodeTest\ Predicate_1 ... Predicate_m$$

   Реализация спецификатора оси при своей работе должна сохранить в контексте столько предков контекстного узла, сколько их требуется для данного шага доступа. Для предикатов не требуется сохранять каких-либо предков, поскольку задачей предикатов является лишь фильтрация контекстного узла, т.е. фактически ответ вида “да-нет” на вопрос, включается ли контекстный узел в результат шага доступа. Шаг доступа должен требовать, чтобы ему предоставили в контексте такое число предков, которое потребуется для вычисления спецификатора оси, с учетом также максимума по числу предков, которые потребуются для применения предикатов к результату выполнения оси. В виде формулы это может быть записано следующим образом:

   $$Step(ancestors\_number) = AxisSpecifier(\max_{i=1,m}(ancestors\_number ; (Predicate_i(0)))).$$

3. **Спецификатор оси** (axis specifier) – эта то правило грамматики XPath, которому уделяется основная роль в рассматриваемом алгоритме. Именно реализация каждой конкретной оси отвечает за то, чтобы выбрать в соответствии с семантикой оси узлы документа (возможно, пользуясь предками контекстного узла, сохраненными в контексте), а также непосредственно отвечает за то, чтобы сохранить в своем результирующем контексте такое количество предков, которое потребуется для дальнейшего вычисления всего выражения языка XPath. Основные принципы реализации осей с учетом сохраненных предков контекстного узла будут рассмотрены в разделе 5.3, а сейчас мы определим количество предков, которое необходимо сохранять для корректной работы каждой оси:

   $$\begin{aligned}
   Ancestor(ancestors_number) &= +\infty \quad ; \\
   Ancestor\text{-}or\text{-}self(ancestors_number) &= +\infty \quad ; \\
   Attribute(ancestors_number) &= \max(ancestors_number - 1; 0) \quad ; \\
   Child(ancestors_number) &= \max(ancestors_number - 1; 0) \quad ; \\
   Descendant(ancestors_number) &= \max(ancestors_number - 1; 0) \quad ; \\
   Descendant\text{-}or\text{-}self(ancestors_number) &= ancestors_number \quad ; \\
   Following(ancestors_number) &= +\infty \quad ; \\
   Following\text{-}sibling(ancestors_number) &= \max(ancestors_number; 1) \quad ; \\
   Namespace(ancestors_number) &= \max(ancestors_number - 1; 0) \quad ; \\
   Parent(ancestors_number) &= ancestors_number + 1 \quad ; \\
   Preceding(ancestors_number) &= +\infty \quad ; \\
   Preceding\text{-}sibling(ancestors_number) &= \max(ancestors_number; 1) \quad ; \\
   Self(ancestors_number) &= ancestors_number \quad .
   \end{aligned}$$

4. **Предикат** (predicate) содержит в себе выражение языка XPath общего вида. Для **выражения** (expr), представляющего собой арифметическую или булевскую операцию или операцию сравнения, для его корректного вычисления в контексте будет требоваться такое количество предков контекстного узла, сколько их по максимуму требуется для каждого из подвыражений данной операции. Если от выражения одного из перечисленных типов требуется сохранять в его результирующем контексте отличное от нуля количество предков, то это свидетельствует о наличии семантической ошибки в анализируемом исходном выражении XPath, потому что в нем происходит применение оси к аргументу, не являющемуся узлом, что некорректно согласно Спецификации XPath [1].
5. **Выражение объединения** (union expression) требует от каждого из своего аргументов сохранить в контексте такое количество предков, которое потребовали от него самого. Выражению объединения должно быть предоставлено в контексте такое количество предков, которое по максимуму потребовалось для вычисления его аргументов:

$$\text{UnionExpr} ::= \text{PathExpr}_1 \mid ... \mid \text{PathExpr}_k \;\; ;$$

$$\text{UnionExpr}(\text{ancestors\_number}) = \max_{i = 1, k} (\text{PathExpr}_i(\text{ancestors\_number})) .$$

6. **Выражение пути** (path expression) по своей структуре схоже с правилом для пути доступа

$$\text{PathExpr} ::= \text{FilterExpr} / \text{Step}_1 / \text{Step}_2 /.../ \text{Step}_n \;\; ;$$

поэтому на него накладываются условия, аналогичные условиям на путь доступа:

$$\text{PathExpr}(\text{ancestors\_number}) = \text{FilterExpr} (\text{Step}_1(\text{Step}_2 (... (\text{Step}_n(\text{ancestors\_number})) ...))) \;\; .$$

Аналогичная взаимосвязь прослеживается и между **выражением фильтрации** (filter expression) и рассмотренным ранее шагом доступа.

$$\text{FilterExpr} ::= \text{PrimaryExpr Predicate}_1 ... \text{Predicate}_p \;\; ;$$

$$\text{FilterExpr}(\text{ancestors\_number}) = \text{PrimaryExpr}(\max(\text{ancestors\_number}) \;\; ; \;\; \max_{I = 1, p} (\text{Predicate}_i(0)))) \;\; .$$

7. **Базовое выражение** (primary expression), являющееся одной из констант, сохранение предков в контексте не требует, т.к. константа сама создает новый контекст. Неправомерным будет также требовать от константы сохранять отличное от нуля количество предков в ее результирующем контексте, потому что подобная ситуация сигнализирует о том, что в исходном анализируемом выражении присутствует применение оси к константе (не являющейся узлом), что является семантической ошибкой выражения.

8. Для базового выражения, представляющего собой вызов одной из функций из Библиотеки базовых функций XPath [9], для большинства из этих функций не требуется сохранения предков ни для ее аргументов, ни для возвращаемого результата. Исключение составляют лишь функция `lang`, требующая сохранения всех предков для своего аргумента; и функция `id`, возвращающая набор узлов, поскольку к нему на последующих шагах доступа могут применяться обратные оси XPath.

5.3. Обоснование алгоритма

Обоснование рассмотренного в предыдущем разделе алгоритма вычисления выражений XPath на основе сохраненных в контексте предков контекстного узла производится следующей теоремой.

Теорема 1. При распределении количества предков по грамматическим правилам языка XPath в соответствии с вышеописанным алгоритмом, вычисление выражения языка XPath может быть построено таким образом, что требуемые для вычисления обратных осей XPath предки контекстного узла всегда могут быть извлечены непосредственно из контекста, без необходимости иметь в дереве документа указатели с дочерних узлов на родительские узлы.

Доказательство 2. Доказательство теоремы проведем в 2 этапа. На первом этапе рассмотрим количество предков, которое требуется для вычисления каждой из осей языка XPath. На втором этапе покажем, что рассмотренное в алгоритме распределение количества предков между грамматическими правилами обеспечит для произвольного выражения языка XPath наличие требуемого количества предков для каждой из осей, встречающейся в этом выражении.

1. Рассмотрим каждую из осей языка XPath и установим количество предков, необходимое для ее вычисления.
   - Ось *parent* по определению выбирает родительский узел для контекстного узла, поэтому для реализации данной оси необходимо иметь в контексте сохраненный родительский узел, что соответствует количеству предков, равному 1.
   - Оси *ancestor* и *ancestor-or-self* по определению выбирают всех предков контекстного узла (ось *ancestor-or-self* выбирает также контекстный узел). В соответствии с предлагаемым подходом вычисления выражений языка XPath реализация этих осей требует для себя хранения в контексте всех предков контекстного узла, что согласно нашему определению из раздела 5.1 соответствует количеству предков, равному $+\infty$.
   - Оси *child*, *descendant*, *attribute* и *namespace* обладают тем общим свойством, что каждая из них осуществляет спуск по дереву документа по крайней мере на глубину 1; поэтому реализация этих осей не требует обращения к предкам контекстного узла. Более того, примечательным для данных осей является то свойство, что

исходный контекстный узел является родителем (а для оси *descendant* – предком) результирующего контекстного узла; следовательно, реализации этих осей при необходимости могут сохранять в контексте количество предков, на 1 большее того количества предков, которое находилось во входном контексте.

- Оси *self* и *descendant-or-self* не требуют обращения к предкам контекстного узла, а также ввиду своей семантики не могут сохранить в контексте количество предков, большее, чем было сохранено во входном контексте.
- При рассмотрении осей *following-sibling* и *preceding-sibling* необходимо заметить, что алгоритм разработан для представления XML-документов в виде SXML, где отсутствуют указатели между соседними узлами-братьями, и поэтому доступ к ним осуществляется через их (общий) родительский узел. В соответствии с таким способом вычисления, реализация данных осей должна требовать в контексте наличие сохраненного родителя контекстного узла, что соответствует количеству предков, равному 1.
- Реализация осей *following* и *preceding* требует в контексте количества предков, равное $+\infty$, поскольку при выборе всех узлов XML-документа, следующих за контекстным узлом (для оси *preceding* – предшествующих контекстному узлу) в порядке документа, необходимо подниматься по дереву документа до корневого узла.

2. На предыдущем этапе доказательства было показано, какое количество предков требуется для каждой из осей XPath для корректной работы в соответствии с предлагаемым способом вычисления обратных осей. Теперь заметим, что распределение количества предков между остальными грамматическими правилами языка XPath построено в алгоритме таким образом, чтобы обеспечить каждую из осей, встречающуюся в некотором выражении XPath, требуемым для нее количеством предков. В справедливости данного утверждения можно убедиться, последовательно анализируя взаимоотношение между грамматическими правилами XPath, начиная от правил, являющихся листовыми вершинами абстрактного синтаксического дерева (т.е. не содержащих внутри себя других правил), и постепенно переходя к более сложным правилам по принципу суперпозиции.

   Рассуждения по поводу количества предков для каждого из грамматических правил XPath приводились при рассмотрении алгоритма. Так, например, было отмечено, что шаги доступа (location steps) в пути доступа (location path) должны рассматриваться в обратном порядке, с той целью, чтобы каждый шаг доступа после своего вычисления сохранял в контексте количество предков, необходимое для вычисления последующих шагов. Аналогичные рассуждения по поводу распределения количества предков для остальных грамматических правил XPath повторяют рассуждения, сделанные при рассмотрении алгоритма, и поэтому здесь опущены.

Поскольку каждая из осей, встречающаяся в произвольном выражении XPath, в соответствии с алгоритмом получает для себя необходимое количество предков, это и означает возможность вычисления любой обратной оси XPath за счет извлечения предков контекстного узла непосредственно из контекста и доказывает утверждение теоремы.

Замечание 2. При доказательстве теоремы попутно приведена схема вычисления для каждой из обратных осей XPath с помощью сохраненных в контексте узлов – предков контекстного узла.

Замечание 3. Сформулированное в алгоритме распределение количества предков между грамматическими правилами языка XPath допускает для некоторых выражений XPath наличие сохраненных предков в контексте, полученном в результате вычисления всего выражения. В качестве примера подобного выражения рассмотрим путь доступа

```
/descendant::tr[parent::table]
```

выбирающий все узлы документа с именем *tr*, родительские узлы которых имеют имя *table*[4]. Легко видеть, что в данном пути доступа от реализации оси *descendant* требуется сохранить в контексте количество предков, равное 1, поскольку внутри предиката используется ось *parent*, требующая для себя иметь в контексте сохраненный родительский узел контекстного узла. После проверки контекста на предмет удовлетворения условию предиката дальнейшее хранение родительского узла в контексте становится ненужным. При практической реализации предложенного алгоритма вычисления обратных осей может быть полезным удалять из контекста хранящихся там предков в конце вычисления выражения.

6. Свойства алгоритма

В данном разделе описываются некоторые свойства предложенного алгоритма, вытекающие из особенностей способа вычисления обратных осей языка XPath.

6.1. Вычисление выражения и сборка мусора

Сборка мусора – это процесс автоматического управления памятью, который производит поиск и освобождение областей памяти, занимаемых теми объектами программы, на которые программа никогда больше не будет ссылаться в будущем. Автоматическая сборка мусора обеспечивается в языках функционального программирования семейства Лисп, и в частности в языке Scheme, на котором написана рассматриваемая в данной работе реализация языка XPath.

[4] Использованная форма записи данного пути доступа выбрана исключительно с целью иллюстрации замечания. Рассуждения о том, что путь доступа с аналогичной семантикой может быть записан по-другому, без использования предиката, мы оставляем за пределами нашего рассмотрения в данной статье.

Предложенный в разделе 5 алгоритм вычисления выражений языка XPath предоставляет сборщику мусора возможность уже *по ходу вычисления* над некоторым SXML-документом выражения XPath освобождать те части документа, которые уже больше не потребуются для дальнейшего вычисления выражения.

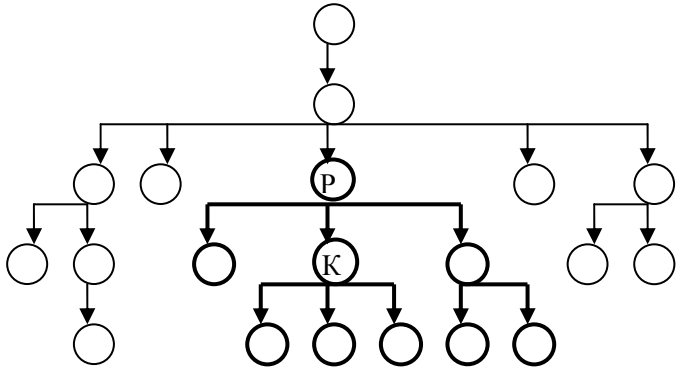

Рис. 3. Пример дерева XML-документа, над которым производится вычисление некоторого выражения XPath. Буквой `K` обозначен контекстный узел, буквой `P` – родительский узел для контекстного узла.

Проиллюстрируем данное утверждение с помощью рис. 3, изображающего дерево некоторого XML-документа, представленного в виде SXML. Необходимо заметить, что в соответствии с реализацией S-выражений между узлами дерева имеются лишь однонаправленные указатели “предок-потомок”, которые на рис. 3 обозначены стрелками, ориентированными в направлении указателей. Предположим, что над рассматриваемым документом производится вычисление некоторого выражения XPath, и в данный момент вычислений контекстным узлом является узел дерева документа, который помечен на рис. 3 буквой `K`. Будем также считать, что для последующего вычисления выражения требуется родительский узел контекстного узла, и поэтому в соответствии с предложенным алгоритмом в контексте вычисления сохранен этот родительский узел (на рис. 3 он помечен буквой `P`).

Проследив за указателями, имеющимися в дереве документа, легко видеть, что дальнейшее вычисление выражения XPath будет производиться в рамках поддерева, выделенного на рис. 3 жирными линиями, поскольку лишь до узлов этого поддерева можно добраться из узлов, содержащихся в рассматриваемом нами контексте вычисления – `K` и `P`. Остальные узлы дерева документа (изображенные на рис. 3 тонкими линиями) в дальнейшем вычислении выражения XPath заведомо участвовать не будут, поэтому, если на них нет ссылок и из других мест прикладной программы, эти узлы уже в процессе вычисления выражения XPath могут освобождаться сборщиком мусора.

Рассмотренное свойство имеет важное значение для случаев, когда вычисление выражения XPath осуществляется над большими деревьями документов, и при этом прикладное приложение интересует лишь *результат* вычисления, который содержит лишь небольшие поддеревья дерева исходного документа.

6.2. Уникальность узлов

Язык XPath часто используется не только как самостоятельный инструмент, но также как неотъемлемая составная часть таких языков как XQuery и XSLT. Консорциум Всемирной Сети изначально разрабатывал язык XPath с целью использования его в других языках платформы XML как инструмента для адресации структурных частей XML-документов.

В то время как язык XPath в силу принципов своего дизайна способен лишь *адресоваться* к узлам XML-документа без возможности их преобразования, язык запросов к XML-документам XQuery [14], использующий XPath в качестве своей составной части, уже позволяет *создавать* новые узлы, что реализуется с помощью наличия в языке XQuery конструкторов для разных типов узлов. Для определения семантики конструируемых узлов в языке XQuery вводится понятие *уникального идентификатора узла* (node identity). Уникальный идентификатор дается каждому узлу XML-документа, над которым производится выполнение запроса XQuery. Каждому сконструированному узлу и каждому его потомку также присваивается свой собственный уникальный идентификатор, не совпадающий ни с одним уникальным идентификатором остальных узлов.

Из описанного выше понятия уникального идентификатора узла следует, что когда в содержимом конструктора элемента присутствует некоторый узел `N` документа, семантика вычисления конструктора такова, как если производится копирование узла `N`, и эта копия становится дочерним узлом для конструируемого элемента. Ввиду того, что в копии узла `N` используется ссылка на другой родительский узел, многие практические реализации XQuery используют прямолинейный способ реализации уникального идентификатора узлов, основанный на копировании узлов, содержащихся внутри конструкторов элементов. Очевидным недостатком подобного прямолинейного подхода является необходимость глубокого копирования поддеревьев документа, требующее больших накладных расходов по занимаемой памяти и времени выполнения.

Проблемы глубокого копирования поддеревьев можно избежать при функциональной реализации языка запросов к XML на основе предлагаемого в данной ра-боте алгоритма вычисления выражений XPath. Поскольку язык функционального программирования Scheme представляет SXML-документ в виде иерархического однонаправленного связного списка, и поскольку функциональная парадигма программирования исключает побочные эффекты вычисления, то поддерево, являющееся общим для нескольких деревьев, автоматически хранится в одной физической копии. В предлагаемом подходе вычисления выражений XPath указатели на родительские узлы моделируются за счет их хранения

в контексте вычисляемого выражения, и поэтому контекст задает то дерево, которому принадлежит контекстный узел и соответствующее ему поддерево.

Достаточно заметить, что конструируемый узел не имеет родительского элемента, и предлагаемый подход вычисления выражений XPath естественным образом расширяется до возможностей языка запросов, обеспечивая поддержку семантики уникальных идентификаторов узлов, без необходимости физи-ческого копирования узлов, содержащихся внутри конструкторов элементов.

6.3. Вычисление выражений конкурентными транзакциями

Поскольку язык XPath используется как составная часть одного из языков внесения модификаций в XML-документы [15], то реализация языка XPath потенциально может рассматриваться как составная часть инструмента по внесению модификаций в XML-документы несколькими конкурентными транзакциями. Данная область применения языка XPath порождает интерес к исследованию условий, при которых возможно гарантировать условно-последовательное (serializable) вычисление выражений XPath конкурентными транзакциями над общим XML-документом.

В [16] показывается, что применение стандартного двухфазного протокола блокирования к данным древовидной структуры, которую имеет XML-документ, приводит к низкому параллелизму выполнения конкурентных транзакций. Для древовидных структур данных в [16] предложен специальный протокол блокирования, который обеспечивает возможность последовательного упорядочения операций нескольких транзакций за счет эксплуатации того факта, что все обращения к элементам древовидной структуры предполагают просмотр дерева в направлении от корневого узла к листовым узлам.

Ниже мы воспроизведем условия **протокола блокирования древовидных структур** (tree-locking protocol). Доказательство корректности данного протокола дается в [16].

1. Первый запрос на блокирование, инициируемый транзакцией, может относится к любому узлу дерева.
2. Последующие запросы на блокирование должны удовлетворяться только в том случае, если транзакция обладает блокировкой узла, родительского по отношению к текущему.
3. Операции разблокирования разрешено выполнять в любые моменты времени.
4. Транзакция не имеет возможности повторного захвата блокировки узла после ее освобождения – даже в том случае, если принадлежащая транзакции блокировка родительского узла всё еще активна.

Необходимо также заметить, что из доказательства корректности протокола блокирования древовидных структур [16] следует, что в том случае, если все транзакции начинают запросы на блокирование с корневого узла дерева – как это происходит в случае работы с XML – то условие 4 протокола можно дополнительно ослабить и разрешить транзакции повторно захватывать узел после освобождения, если транзакция сохранила блокировку родительского узла.

Заметим, что условия протокола блокирования древовидных структур очень хорошо ложатся на предлагаемый в данной работе способ вычисления выражений языка XPath. Действительно, хранение в контексте выражения предков контекстного узла представляет собой удержание блокировки на эти узлы. Когда некоторый узел – предок контекстного узла – перестает храниться в контексте, это можно рассматривать как снятие блокировки с данного узла-предка. Как описывалось в разделе 5, предлагаемый способ вычисления выражений XPath построен таким образом, что необходимые для вычисления выражения предки контекстного узла сохранятся в контексте, продолжают храниться там в течение всего времени, пока ожидается их использование, и не запрашиваются повторно, будучи однажды освобождены из контекста.

Реализация языка XPath, основанная на предлагаемом подходе вычисления обратных осей, может естественным образом обеспечить поддержку протокола блокирования древовидных структур и таким образом предоставить возможность работы с XML-документами конкурентным транзакциям.

7. Ограничения алгоритма

Для некоторых выражений языка XPath возможно их вычисление с использованием меньшего количества предков, чем это предписывается рассмотренным алгоритмом. В качестве примера рассмотрим приведенный ниже шаг доступа XPath (который может выступать в качестве полноправного выражения XPath или быть частью более сложного выражения):

```
ancestor::*[position()<3]
```

Нетрудно видеть, что в соответствии со Спецификацией XPath [1] данный шаг доступа выбирает всех тех предков контекстного узла, контекстная позиция которых меньше 3; т.е., другими словами, родителя и прародителя контекстного узла. Рассматривая данный шаг доступа как единое целое, представляется разумным, чтобы он требовал для себя количество предков, равное 2, поскольку предки контекстного узла, имеющие контекстную позицию, большую 2, отсекаются предикатом. Однако предлагаемый алгоритм вычисления выражений XPath строит свою работу по принципу анализа подвыражений, входящих в состав анализируемого выражения. В данном примере, обнаружив спецификатор оси `ancestor`, входящей в состав шага доступа, алгоритм принимает решение о необходимости сохранения в контексте всех предков контекстного узла. Алгоритм не распознает семантическую зависимость, существующую в данном примере между спецификатором оси и предикатом в отношении количества предков.

Выявление на фазе статического анализа выражения зависимостей между его подвыражениями, аналогичных показанным в данном примере, является более общей задачей оптимизации вычисления выражений языка XPath.

Предложенный в данной работе алгоритм ориентировался на подмножество способов оптимизации – оптимизацию вычисления обратных осей языка XPath, – и поэтому использовал ограниченные возможности статического анализа рассматриваемого выражения.

8. Эксперименты

Рассмотренный алгоритм вычисления выражений языка XPath был полностью реализован в виде расширения к системе SXPath, предоставляющей возможности языка запросов к SXML-документам. Для определения количественного эффекта, достигаемого благодаря оптимизации вычисления обратных осей языка XPath, были проведены эксперименты по замерам производительности. В экспериментах сравнивалась реализация SXPath “до оптимизации”, в которой вычисление обратных осей XPath производилось за счет поиска необходимых узлов от корня документа, и реализованное расширение к SXPath, в котором выражения XPath вычисляются с помощью предложенного в работе алгоритма.

В качестве тестовых SXML-документов для экспериментов использовались автоматически сгенерированные документы различной глубины, элементы в каждом документе образуют сбалансированное дерево. Пример используемого в эксперименте SXML-документа для глубины 4 приведен на рис. 4. У каждого элемента в документе есть дочерний текстовый узел. Также каждый элемент за исключением элементов самого нижнего уровня имеет по 2 дочерних элемента, т.е. элементы документа образуют сбалансированное двоичное дерево. Имена всех элементов в документе различны, также различны значения всех текстовых узлов.

```
(*TOP*
 (elem1
  (elem2
    (elem3 (elem4 "text5") (elem6 "text7") "text8")
    (elem9 (elem10 "text11") (elem12 "text13") "text14")
    "text15")
  (elem16
    (elem17 (elem18 "text19") (elem20 "text21") "text22")
    (elem23 (elem24 "text25") (elem26 "text27") "text28")
    "text29")
  "text30"))
```

Рис. 4. Тестовый SXML-документ глубины 4.

В качестве тестовых выражений XPath использовались случайным образом сгенерированные пути доступа, состоящие из фиксированного числа шагов доступа. Каждый из шагов доступа использует произвольную из осей языка XPath, один из тестов узлов `node()`, `*` или `text()`, и не содержит предикатов. Пример одного из сгенерированных тестовых путей доступа, состоящего из 4 шагов доступа, приведен на рис. 5.


```
descendant::*/following-sibling::node()/self::text()/parent::*
```

Рис. 5. Пример тестового пути доступа, состоящего из 4 шагов доступа.

В таблицах на рисунках 6 и 7 приведены результаты временных затрат на вычисление путей доступа, состоящих из 3 и 4 шагов доступа соответственно. В строках каждой таблицы указаны глубины тестовых SXML-документов, над которыми проводилось вычисление путей доступа. В результаты эксперимента вошли только те из случайно сгенерированных путей доступа, результатом вычисления которых являлось непустое множество узлов документа. Каждый из таких нетривиальных путей доступа вычислялся при помощи неоптимизированной реализации SXPath, в которой вычисление обратных осей XPath производится за счет поиска необходимых узлов от корня документа, и при помощи реализованного расширения SXPath, в котором выражения XPath вычисляются с помощью предложенного в работе алгоритма. В ячейках таблиц на рисунках 6 и 7 показаны усредненные времена вычисления, полученные для нескольких (по 20 для каждой строки) случайных путей доступа. Данные в таблицах не включают в себя время, требуемое для разбора путей доступа, и поэтому более точно отражают время вычисления над тестовыми документами.

Глубина дерева документа	Время вычисления, сек	
	Поиск предков контекстного узла от корня документа	Предложенный в работе алгоритм вычисления обратных осей
4	0.004	0.003
5	0.055	0.010
6	0.105	0.020
7	0.303	0.049
8	1.870	0.632
9	10.862	3.874
10	25.008	8.503

Рис. 6. Результаты изменений для путей доступа XPath, состоящих из 3 шагов доступа.

Глубина дерева документа	Время вычисления, сек	
	Поиск предков контекстного узла от корня документа	Предложенный в работе алгоритм вычисления обратных осей
4	0.022	0.012
5	0.094	0.020
6	0.145	0.027
7	0.552	0.066
8	17.465	3.992
9	29.642	6.987
10	128.570	33.958

Рис. 7. Результаты изменений для путей доступа XPath, состоящих из 4 шагов доступа.

Из таблиц легко видеть, что среднее время вычисления выражений XPath с использованием предложенного в работе алгоритма всегда меньше среднего времени, необходимого для вычисления тех же выражений, когда обратные оси вычисляются за счет поиска предков контекстного узла от корня документа. Относительный разрыв во времени увеличивается при увеличении глубины рассматриваемого тестового документа, т.е. при увеличении числа узлов в нем.

В ходе дополнительных экспериментов также было установлено, что накладные расходы предложенного алгоритма при вычислении выражений XPath, не содержащих обратных осей, незначительны, что достигается за счет описанного в разделе 5 распределения количества предков по грамматическим правилам языка XPath, минимизирующего количество хранимых в контексте предков контекстного узла.

9. Заключение

В работе решалась задача оптимизации вычисления обратных осей языка XPath функциональными методами и преодоление проблемы отсутствия в SXML указателей с дочерних узлов на родительские узлы.

Был проведен обзор родственных работ по предметной области, посвященных как вопросам восстановления указателей на родительские узлы в дереве SXML-документа, так и вопросам оптимизации вычисления выражений XPath. Обсуждался контекст вычисления XPath и хранимая в контексте информация. Был предложен термин **количество предков**, позволяющий однозначно задавать те узлы – предков контекстного узла, – которые необходимо

сохранить внутри контекста. Было составлено распределение количества предков по подвыражениям выражения XPath, основанное на грамматике XPath и позволяющее минимизировать количество хранимых в контексте предков контекстного узла, необходимых для вычисления данного подвыражения. Было показано, как каждая из обратных осей XPath может быть вычислена при наличии в контексте необходимого количества предков контекстного узла, без необходимости иметь явные указатели с дочерних узлов на родительские узлы в дереве документа.

При обосновании алгоритма было доказано, что с помощью предлагаемого подхода произвольное выражение языка XPath может быть вычислено даже при отсутствии в дереве документа указателей с дочерних узлов на родительские узлы. Рассматривались свойства предложенного алгоритма и присущие алгоритму ограничения. Были проведены эксперименты, подтвердившие, что предложенный алгоритм позволяет оптимизировать вычисление обратных осей языка XPath над SXML-документами по сравнению с используемым до этого способом их вычисления в реализации языка XPath функциональными методами.

Полученные в работе результаты восстанавливают изоморфизм между SXML и Моделью Данных XPath и подтверждают, что SXML является полной моделью Информационного Пространства XML.

Литература

1. J. Clark и S. DeRose (редакторы). Язык XML Path (XPath) Версия 1.0. Рекомендация W3C от 16 Ноября 1999. http://citforum.ru/internet/xpath/index.shtml
2. Лизоркин Д.А. и Лисовский К.Ю. SXML: XML-документ как S-выражение. Электронные библиотеки, 2003, Том 6, Выпуск 2. http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2003/part2/LK
3. O. Kiselyov and K. Lisovsky. XML, XPath, XSLT Implementation as SXML, SXPath and SXSLT. International Lisp Conference ILC 2002, San Francisco. October, 2002. http://www.okmij.org/ftp/papers/SXs.pdf
4. R. Kelsey, W. Clinger and J. Rees (editors). Revised(5) Report on the Algorithmic Language Scheme. Higher-Order and Symbolic Computation, Vol. 11, No. 1, August 1998, and ACM SIGPLAN Notices, Vol. 33, No. 9, September 1998. http://www.schemers.org/Documents/ Standards/R5RS/r5rs.pdf
5. Лисовский К. Ю. Разработка XML-приложений на языке Scheme. Программирование, выпуск 28, номер 4, 2002. http://www.maik.rssi.ru/journals/procom.htm
6. J. Cowan and R. Tobin (editors). XML Information Set (Second Edition). W3C Recommendation 4 February 2004. http://www.w3.org/TR/xml-infoset/
7. O. Kiselyov. On parent pointers in SXML trees. http://www.okmij.org/ftp/Scheme/parent-pointers.txt
8. A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie and J. Simeon (editors). XML Path Language (XPath) 2.0 W3C Working Draft 23 July 2004. http://www.w3.org/TR/2004/WD-xpath20-20040723
9. Когаловский М.Р. Глоссарий по технологиям платформы XML. Версия 4 (25-11-2003). http://www.elbib.ru/index.phtml?page=elbib/rus/methodology/xmlbase/glossary_XML

10. Лизоркин Д.А. и Лисовский К.Ю. Язык XML Path (XPath) и его функциональная реализация SXPath. Электронные Библиотеки, 2003, Том 6, Выпуск 4. http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2003/part4/LL
11. O. Kiselyov. SXML, revision 3.0, March 12, 2004. http://okmij.org/ftp/Scheme/SXML.html
12. K. Lisovsky. STX: Scheme-enabled XSLT processor. http://www.pair.com/lisovsky/transform/stx/
13. D. Olteanu, H. Meuss, T. Furche and F. Bry. XPath: Looking Forward. Proc. of the EDBT Workshop on XML Data Management (XMLDM), 2002. http://www.csd.uch.gr/ hy561/Papers/looking_forward.pdf
14. S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie and J. Simeon (editors). XQuery 1.0: An XML Query Language. W3C Working Draft, 12 November 2003. http://www.w3.org/TR/2003/WD-xquery-20031112/
15. P. Lehti. Design and Implementation of a Data Manipulation Processor for a XML Query Language. Ph.D. thesis. Technische Universitat Darmstadt, August 2001. http://www.ipsi.fraunhofer.de/~ lehti/
16. Гектор Гарсиа-Молина, Джеффри Д. Ульман и Дженнифер Уидом. Системы Баз Данных. Полный курс. Пер. с англ. – Москва, Издательский дом "Вильямс", 2003. ISBN 5-8459-0384-X (рус.)