УДК 004.432.4

XLog - язык обработки HTML и XML документов

Магазов С.С., Чугунов Д.О.

МГТУ им. Н.Э. Баумана
2006mag@mail.ru, d.chugunov11@mail.ru

Аннотация. В работе дается описание основных идей декларативного языка XLog. XLog предназначен для разработки систем интеллектуальной обработки Internet-информации. Приведен синтаксис и семантика основных конструкций языка, рассмотрены примеры. Приведено краткое описание архитектуры интерпретатора.
Ключевые слова: декларативный язык, XML/HTML-документ, обработка текстов, Internet, интерпретатор.

Abstract. This paper presents the declarative language XLog. XLog is designed for the development of intelligent Internet information processing systems. The paper contains syntax and semantics of the basic language constructs and examples. We give the brief description of the architecture of the interpreter.
Keywords: declarative language, XML/HTML-document, word processing, Internet, interpreter.

Введение

Аналитики Cisco прогнозируют, что к 2015 году Internet-трафик документов в формате HTML достигнет 245 терабайт в секунду. Постепенно складываются новые реалии в Internet-сообществе, появляются совершенно новые требования к обработке информации. IT-компании все больше уделяют внимание разработке интеллектуальных систем обработки информации. На рынке появляются технологии, ориентированные на разработку таких систем. На настоящий момент разработаны языки XPath, XQuery, XSLT [4,5].

Язык XPath предназначен для навигации по XML-документам. Язык XQuery, используя конструкции XPath, позволяет организовывать алгоритмы обработки XML-документов [2]. Язык XSLT позволяет преобразовывать XML-документы в формат HTML.

Разработчики языка XLog ставили перед собой цель создания декларативного языка для анализа и синтеза XML и HTML-документов. Под анализом понимается извлечение частей XML/HTML-документа и их лингвистическая обработка. Под синтезом понимается преобразование XML/HTML-документа в XML/HTML-документ или структурирование текстовой информации в виде XML/HTML-документа.

XLog предназначен для разработки интеллектуальных систем обработки XLog-документов. XLog-документ являются расширением XML/HTML формата, в нем допускается использование переменных. Синтаксис языка XLog похож на синтаксис языка Prolog, тело программы состоит из разделов: правила и цель. Но есть и существенные различия, например, разработан оригинальный механизм унификации XLog-документов. Для анализа и синтеза XML/HTML-документов адаптированы механизмы возврата и рекурсии. Синтаксис функций и операторов стал более лаконичным. В XLog включены адаптированные конструкции языков XPath, XQuery и XSLT. XLog имеет механизмы интеграции с web-серверами и

базами данных. В статье сформулированы основные идеи языка XLog и дается краткий обзор конструкций языка.

1. XLog-документ

Множество *XLog-документов* включает в себя стандартные XML/HTML-документы и XML/HTML-документы с переменными, которые назовем *шаблонами*. В частности, XLog-документ может состоять из одной переменной. Переменные, используемые в XLog-документе, бывают двух сортов: *атрибутивная переменная*, принимающая в качестве значений атрибуты XLog-документа, и *элементная переменная*, принимающая в качестве значений XLog-элемент или список XLog-элементов. Определение *атрибута* аналогично определению атрибута в XML-документе. XLog-элемент понимается так же как элемент в XML-документе, то есть он состоит из открывающего и закрывающего тегов, обрамляющих текст или другие элементы [4].

Формат переменных следующий:

$identifier – атрибутивная переменная;

$_ – безымянная атрибутивная переменная;

$$identifier – элементная переменная;

$$_ – безымянная элементная переменная для элементов XLog-документов,

где *identifier* – идентификатор. Безымянная переменная используется так же, как и именная переменная, но к ней нельзя обратиться из программы.

Во время исполнения программы атрибутивная переменная по стандарту XML может принимать строковое значение.

Элементная переменная может принимать значения следующих базовых типов: числовой (вещественный или целочисленный), строковый, XLog-документ, список XLog-документов.

Список XLog-документов задаётся следующим образом: <$ob_1 \ldots ob_n$> или <$ob_1 \ldots ob_n$ |$$X >, где ob_i – либо объект одного из базовых типов, либо список, либо переменная. $$X – элементная переменная.

Атрибутивная переменная может стоять на месте значений атрибутов, Элементная переменная может стоять на месте элементов XLog-документа.

Пример 1.1: Предметом нашего анализа будет HTML-документ, состоящий из таблицы. Таблица имеет три столбца с текстовой информацией.

```
<html>
<head> <title> Пример </title> </head>
<body>
   <table border="1">
      <tr>
         <td>   </td>
         <td>Петя послушный мальчик. Петя пошел в школу</td>
         <td>   </td>
      </tr>
   </table>
</body>
</html>
```

В результанте анализа этого HTML-документа должна быть выделена текстовая информация и произведена простая классификация предложений. Для анализа можно использовать следующий шаблон:

```
<html>
$$_
<body>
   <table>
      <tr>
         <td>$$X1</td>
         <td>$$X2</td>
         <td>$$X3</td>
      </tr>
   </table>
</body>
</html>
```

2. Язык XLog

Здесь будет дано описание основных конструкций языка XLog, а также приводится структура XLog программы.

2.1 Точечные операторы

В реализацию XLog входит библиотека точечных операторов, которая может неограниченно расширяться. Точечные операторы, предназначены для обработки текстовой информации. Особенностями текстовой информации является наличие большого числа атрибутов. Поэтому операторы обеспечивающие работу с текстовой информацией должны иметь большой набор аргументов. Аргументы в зависимости от задачи могут быть входными и выходными. Аргументы могут принимать в качестве значения множества элементов XLog документа. Как решение обозначенных проблем предлагается конструкция *точечного оператора*, существенно отличающаяся от «классической» конструкции оператора вида $F(x_1 \ldots x_n)$. Формат точечного оператора следующий:

oper.{*param*=*var* | *var*=*param* | *param*==*var* |
param=*const* | *param*==*const* | *var* | *const*},

где

oper – имя оператора;

param – имя параметра;

var – переменная любо типа;

const – константа базового типа или XLog-документ без переменных.

Запись *param*=*var* или *param*=*const* означает, что входному параметру *param* присваивается значение переменной или константы; запись *var*=*param* означает присваивание значения выходного параметра *param* переменной. Кроме того, синтаксис точечного оператора предусматривает запись сравнений: *param*==*var* означает равенство параметра и значения переменной, а *param*==*const* – равенство параметра и константы. Параметры *var* и *const* называются неименованными.

Порядок именованных параметров в операторе не имеет значения. Оператор в результате исполнения может принимать значение *истина* или *ложь*. При этом выходные параметры принимают значения базовых типов.

Выполнение точечного оператора начинается с присвоения значений всем входным параметрам. Затем возможны два рассматриваемых ниже случая.

1) При заданных значениях входных параметров оператор не может быть вычислен, тогда значения переменных не изменяются, и оператор ложен.

2) Если исполнение оператора возможно, то формируются значения выходных параметров, и проверяются сравнения. Если все сравнения истинны, то оператор истинен и изменяются значения переменных в выражениях вида *var=param*, иначе оператор ложен и значения переменных не изменяются.

Далее рассмотрим точечные операторы Split, Exist Print, isNull, nWord, vWord, aWord, Preposition, предназначенные для обработки списков, строк и текстовой информации. Отметим, что в программной реализации XLog предусмотрен механизм, позволяющий неограниченно расширять набор операторов.

Оператор **Split** разбивает строку на части по разделителю и имеет входные параметры *str*, *delim*, задающие разбиваемую строку и разделитель соответственно, и выходной параметр *val*, в котором возвращается вычисляемый оператором список подстрок.

Пример 2.1.1. Разбиение текста на предложения:

Split.str='Мама мыла раму. Раму мыла мама.'.delim='.'.$X =val

Оператор **Exist** возвращает *true*, если набор, задаваемый путем, не пуст. В противном случае она возвращает *false*. Операция имеет единственный входной неименованный параметр, указывающий путь.

Пример 4. Пусть переменная $X определяет список элементов <a>Element1, <a>Element2, тогда выражение exist.$X\b будет ложным, а выражение exist.$X* – истинным.

Оператор **Print** возвращает всегда *true.* Print имеет произвольный набор неименованных параметров, которые могут быть переменными или константами. Оператор выдает на печать содержимое переменных как строки.

Оператор **isNull** имеет неименованный параметр. Если переменная неконкретезированная, то оператор истинен.

Оператор **nWord** определяет атрибуты существительного. Он имеет следующие входные/выходные параметры: *str*- строка с анализируемым словом; *Gend* – род; *Case* – падеж; *Number* – число.

Оператор **vWord** определяет атрибуты глагола. Он имеет следующие входные/выходные параметры: *str*- строка с анализируемым словом; *time* – время, *Gend* – род, *Number* – число, *Transitional* – переходной.

Оператор **aWord** определяет атрибуты прилагательного. Он имеет следующие входные/выходные параметры: *str* – строка с анализируемым словом, *Gend* – род, *Case* – падеж, *Number* – число.

Оператор **Preposition** определяет атрибуты предлога. Он имеет следующие входные/выходные параметры: *str* – строка с анализируемым словом, *type* – тип отношения, задаваемый предлогом (space, time, causative, и т.д.).

Несколько отличный синтаксис имеет *оператор поиска* (**~**). Этот оператор подробно будет рассмотрен в п. 3. Здесь мы определим только синтаксис оператора. Формат оператора следующий:

~ . *xlog_tmpl* . $variable_1,\ldots, variable_n$, где

xlog_tmpl – это либо URL, либо *xlog*-документ, который задаётся как *@"xlog"*, либо список;

$variable_1,\ldots, variable_n$ – список (возможно, пустой) переменных, которые могут как входить, так и не входить в *xlog_tmpl*.

2. Оператор навигации по XLog-документу

Любая технология обработки XML/HTML-документов должна включать механизмы доступа к частям XML/HTML-документа. Язык XPath [3] решает этот вопрос при помощи механизма путей. В XLog включены механизмы из XPath.

Применяя оператор пути к XLog-документу или набору XLog-элементов, получаем *результирующий список* XLog-элементов.

Путь в XLog имеет следующий вид:

path:=(*var*|*URL)*(\|\\)*ident*$_1$ [*[predicate]*] ... (\|\\)*ident*$_n$[*[predicate]*], где

var – элементная переменная, которая на момент применения оператора должна иметь в качестве значения XLog-документ, либо список XLog-элементов;

URL – путь, указывающий расположение XLog-документа;

ident$_i$ – имя тега;

predicate – предикат, представляющий собой либо логическое выражение, которое вычисляется для каждого элемента, полученного на текущем шаге, либо индекс элемента списка.

Результирующий список на каждом шаге формируется согласно двум правилам, приведенным ниже.

1) Если в текущей позиции пути стоит «\», то имя тега *(ident*$_i$*)*, которое записано после «\», определяет дочерние теги текущего списка (список, полученный на предыдущем шаге), которые могут быть включены в результирующий список.

Если после *ident*$_i$ стоит *[predicate]*, то перед добавлением элемента в результирующий список производится вычисление значения предиката, и элемент добавляется в результирующий список, только если результат равен *true*.

Если в квадратных скобках указан индекс возвращаемого элемента, то в результирующий список будет помещен элемент с указанным номером (нумерация начинается с 1).

2) Если в текущей позиции пути присутствует «\\», то выбираются все потомки элемента текущего списка (список, полученного на предыдущем шаге) с проверкой предиката на истинность.

Как видно из определения, правила формирования результирующего списка похожи на правила XPath.

2.3. Выражения XLog

Помимо операторов в XLog включен ряд *выражений*. Рассмотрим наиболее важные из них.

Выражение **is** является аналогом оператора присваивания. Формат этого выражения следующий:

Var ***is*** *Path*,

где *Path* – путь или выражение, *Var* – переменная.

Так как у нас имеется два типа переменных, то оператор is вынужден выполнять проверку типов: если присваивание невозможно, возникает ошибка времени выполнения.

При помощи выражения **in** определяется принадлежность значения переменной *Var* списку элементов, который определяется путем (*path*). Формат выражения следующий:

Var **in** *Path*,

где *Path* – путь, *Var* – переменная.

Результатом этой операции будет **true**, если результат вычисления выражения принадлежит набору элементов, задаваемых путем, и **false** – в противном случае.

Пример 2.3.1. Пусть переменная $X определяет список элементов:

<a>Element1,

<a>Element2.

Тогда выражение "<a>Element1" **in** $X будет истинным, а выражение "<a>Element6" **in** $X – ложным.

Рассмотрим ряд выражений XLog , подобных конструкциям из XPath, XQuery и XSLT. Синтаксис по возможности сохранен.

Цикл **for** позволяет организовывать просмотр набора элементов и формировать список элементов. Формат **for** следующий:

for var_1 **in** $path_1$ **where** *bool* **return** $path_2$ **to** var_2,

где

var_1, var_2 – переменные;

$path_1$, $path_2$ – пути;

bool – булевское выражение.

На каждом шаге **for** последовательно связывает переменную var_1 с элементами из набора, который задан $path_1$. Далее проверяется булевское выражение *bool*. Если оно истинное, то в набор var_2 добавляются элементы, которые определяются путем $path_2$. Переменная var_1 доступна только внутри цикла.

После исполнения управление передается следующей предикатной конструкции. В случае~~,~~ если $path_1$ – пустое множество, ~~то~~ выражение **for** пропускается~~,~~ и управление передается следующей предикатной конструкции.

Пример 2.3.2. Пусть переменная **$X** определяет набор элементов:

<a><d>Hello1</d>,

Hello2,

<a><c>Hello3</c>,

<c>Hello4</c>.

Тогда в результате выполнения выражения

for $Y **in** $X **where exist**($Y\b**) return** $Y\b **to** $result,

в переменной **$result** будут находиться следующие элементы:

Hello2,

Hello4.

Выражение **if** имеет следующий вид:

if (*bool***) then** $predicate_1$ **else** $predicate_2$

Семантика **if** отличается от общепринятой. Выражение **if** вычисляет значение логического выражения. Если результатом является *true*, то вычисляется

predicate$_1$, в противном случае – *predicate*$_2$. Выражение **if** является истинным только в том случае, если истинным является предикат, которому передано управление (*predicate*$_1$ или *predicate*$_2$). Если выражение **if** истинно, то вычисляется следующая конструкция, иначе — запускается механизм возврата, с которым мы познакомимся в п.4.

В XLog включен оператор отсечения, который используется для прерывания возврата. Отсечение обозначается знаком “!”. Действует отсечение просто: через него невозможно совершить возврат.

2.4 Структура программы

Структура программы на языке XLog похожа на структуру программы на языке PROLOG [1]. Программа состоит из раздела *XLog-правил* (Rule) и раздела *XLog-цели* (Goal). Приведем шаблон программы:

Program *name*

Rule

head:- *predicate_construction,… ;*

…

head:- *predicate_construction,… ;*

head;

Goal

predicate_construction, …;

end

В шаблоне *name* – это имя программы, *head* – заголовок правила, *predicate_construction* – либо точечный оператор, либо конструкция, либо оператор поиска правила.

Заголовок правила имеет три формы.

1) ~(*xlog_tmpl*, *variable*$_{1,…}$, *variable*$_n$), где *xlog_tmpl* – URL, указывающий на место расположения xlog-документа, а *variable*$_{1,…}$, *variable*$_N$ – переменные, которые могут как входить, так и не входить в *xlog_tmpl*. При этом список переменных может вообще отсутствовать в заголовке.

2) ~(@ "*xlog*", *variable*$_1$,…, *variable*$_n$). *Эта форма аналогична первой форме. Различие заключается лишь в том, что в первом аргументе задается не URL, а XLOG-документ.*

3) ~(L, *variable*$_1$,…, *variable*$_n$), где L – список, а *variable*$_i$ – переменные.

Телом правила назовем последовательность *predicate_construction*, входящих в правило. Областью видимости определенных в правиле переменных, т.е. местом в программе, где можно их использовать, является тело правила. Правило может состоять только из заголовка.

Пример 2.4.1. Приведенная ниже программа выделяет из HTML-документа текстовую информацию. В тексте распознаются упрощенные версии атрибутивных предложений (формула предложения *объект-предикат*) и двусоставные предложения. *Двусоставное предложение* содержит два главных члена – подлежащее и сказуемое, причем как подлежащее, так и сказуемое могут быть составными.

```
Program MineText
Rule
// Правило 1.
~('C:\XLog\tmpl.xml', $$X1,$$X2,$$X3) :-
        // в $$Y1 список предложений из первого столбца.
        Split.str=$$X1.delim='.'.$$Y1 =val, ~.$$Y1,
        //в $$Y2 список предложений из второго столбца..
        Split.str=$$X2.delim='.'.$$Y2 =val,~.$$Y2,
        //в $$Y3 список предложений из третьего столбца..
        Split.str=$$X3.delim='.'.$$Y3 =val, ~.$$Y3;
// Правило 2. Выделение предложений и их анализ.
~(<$$X1|$$X2>) :- ~.<sAnl, $$X1>, ~.$$X2;
// Правило 3. Обнаружение конца списка предложений.
~(<>) :-!;
// Правило 4. Преобразование предложения в список слов и проверка,
// является ли предложение атрибутивным
```

```
~(<sAnl, $$X>, $$X):- Split.str=$$X.delim=’ ’.$$Y=val, ~.<AtrSnt,$$Y>;
// Правило 5. Проверка, является ли предложение двухсоставным.
~(<sAnl, $$X>, $$X):- Split.str=$$X.delim=’ ’.$$Y=val,
        ~.<TowPartSnt,$$Y>;
// Правило 6.
~(<AtrSnt,<$$Y1,$$Y2>>):-
        nWord.str=$$Y1.case=’nominative’,aWord.$$Y2,
        print.”атрибутивное предложение”;
// Правило 7.
~(<AtrSnt,<$$Y1,$$Y2,$$Y3>>):- nWord.str=$$Y1.case=’nominative’,
        aWord.$$Y2, nWord.str=$$Y3.case=’nominative’,
        print.”атрибутивное предложение”;
// Правило 8.
~(<TowPartSnt, <$$Y1,$$Y2>>) :-nWord.str=$$Y1.case=’nominative’,
        vWord.$$Y2, print.” простое двусоставное предложение”;
// Правило 9.
~(<TowPartSnt,<$$Y1,$$Y2,$$Y3,$$Y4 >>):-
        nWord.str=$$Y1.case=’nominative’, aWord.$$Y2,
        Preposition.$$Y3, nWord.str=$$Y4.case=’accusative’,
        print.’двусоставные предложения’;
Goal
~.“www.XLog.ru\mysite.html”;
end
```

Исходный код программы имеет небольшой объем (девять правил). Правила определяют термины, хорошо знакомые лингвистам.

3. Унификация

В XLog для организации эффективного доступа к частям XLog-документа, помимо оператора пути добавлен оригинальный и эффективный механизм унификации.

3.1. Унификация XLog-документов

Определим, бинарный оператор *унификации* на XLog-документах и обозначим его как U(H1,H2), где H1, H2 – XLog-документы. Операция унификации устанавливает структурную схожесть двух XLog-документов. В случае успешной унификации переменные, входящие в H1 и H2, могут *конкретизироваться,* то есть им присваиваются в качестве значений фрагменты XLog-документа. Если переменой не присвоено значение, назовем ее *неконкретизированной*. В процессе унификации переменные документов *связываются*. Для связанных переменных справедливо следующее утверждение: при изменении значения какой либо одной из связанных переменных все остальные связанные с ней переменные принимают новое значение.

Дадим индуктивное определение оператору унификации.

1) Если H1, H2 – два одинаковых как строки документа, то они унифицируемы.

2) Пусть даны два заголовка тегов H1=<t1 a_1=V_1 … a_n=V_n> и H2=<t2 b_1=L_1 … b_m=L_m>, где t1 и t2-имена тегов, a_i и b_i – атрибуты тегов.
H1 и H2 унифицируемы, тогда и только, тогда, когда t1=t2 и унификация атрибутов успешна. Рассматриваются атрибуты, которые входят одновременно в оба тега. Остальные атрибуты не оказывают никакого влияния на унификацию.

3) Атрибуты a=V, b=L такие, что a=b, унифицируются, если выполняются условия пунктов 3a–3e.

3a) L и V – константные значения, и L=V.

3b) L и V – несвязанные и не-конкретизированные переменные. В этом случае L и V становятся связанными между собой.

3c) L – связанная конкретизированная переменная, и V – несвязанная переменная. В этом случае L и V становится связанными, а переменная V конкретизируется значением из L.

3d) L – связанная конкретизированная переменная, V – константное значение и значение V совпадает с L.

3e) L – связанная конкретизированная переменная, V – связанная конкретизированная переменная и значение V совпадает L. Переменные L и V становятся связанными.

К пунктам 3a–3e добавляются пункты, в которых L и V меняются местами.

4) Унификация XLog-элемента и переменной осуществляется согласно правилам 4a–4b.

4a) Если H1 – XLog-элемент, и $$X - неконкретизированная переменная, то H1 унифицируется с $$X и $$X принимает значение H1.

4b) Если H1 – XLog-элемент и $$X – конкретизированная переменная, то унификация успешна только в том случае, если значение переменной $$X унифицируется с H1.

5) Унификация элементных переменных. Пусть $$X и $$Y – элементные переменные, тогда унификация выполняется по правилам 5a–5d.

5a) Если переменные $$X и $$Y связаны друг с другом, то унификация успешна.

5b) Если переменная $$X конкретизирована значением A, а переменная $$Y конкретизирована значением B, тогда унификация успешна в случае, если A и B унифицируются.

5c) Если переменная $$X конкретизирована значением A, а переменная $$Y не конкретизирована, то унификация успешна и $$Y примет значение A и станет связана с $$X;

5d) Если переменные $$X и $$Y не конкретизированы и не связаны, то унификация успешна и $$X, $$Y станут связанными.

6) Унификация XLog-элементов выполняется по правилам 6a–6c.

6a) Пусть заданы XLog-элементы

<t1 $a_1=V_1 \ldots a_n=V_n>H_1 \ldots H_p$</t1> ,

<t2 $b_1=L_1 \ldots b_m=L_m>A_1 \ldots A_p$ </t2>.

Они успешно унифицируются, если $H_1 \ldots H_p$ и $A_1 \ldots A_p$, попарно унифицируются и заголовок <t1 $a_1=V_1 \ldots a_n=V_n$> унифицируется с заголовком <t2 $b_1=L_1 \ldots b_m=L_m$>.

6b) Пусть заданы XLog-элементы

A=<t1 $a_1=V_1 \ldots a_n=V_n$> $H_1 \ldots H_p$|$$X </t1>

B= <t2 $b_1=L_1 \ldots b_m=L_m$> $A_1 \ldots A_k$ </t2>

и $p \le k$, тогда A и B унифицируются, если заголовок <t1 $a_1=V_1 \ldots a_n=V_n$> унифицируется с <t2 $b_1=L_1 \ldots b_m=L_m$>, а также $H_1 \ldots H_p$ и $A_1 \ldots A_p$ попарно унифицируются, причём если $$X не конкретизирована, то она принимает значение $A_{p+1} \ldots A_k$, а если $$X конкретизирована, то значение должно унифицироваться с $A_{p+1} \ldots A_k$.

6c) Пусть заданы XLog-элементы

A= <t1 $a_1=V_1 \ldots a_n=V_n$>$H_1 \ldots H_p$|$X</t1>

B= <t2 $b_1=L_1 \ldots b_m=L_m$>$A_1 \ldots A_p$|$Y</t2>,

Тогда A и B унифицируются, если заголовки унифицируется, $H_1 \ldots H_p$ и $A_1 \ldots A_p$ попарно унифицируются и $X унифицируется с $Y.

Пример 2: В результате унификации HTML-документа и шаблона из примера 1 получим: $X1 связана с пустой строкой; $X2 связана со значением «Петя послушный мальчик. Петя пошел в школу»; Переменная $X3 связана с пустой строкой.

3.2 Унификация списков

Списки <$b_1 \ldots b_n$> и <$c_1 \ldots c_n$ > унифицируются, если $U(b_i,c_i)$ для всех $i \le n$.

Списки <$b_1 \ldots b_n$> и <$c_1 \ldots c_k$ |$$X >, где $k<n$, унифицируются, если $U(b_i,c_i)$ для всех $i \le k$. Переменная $$X конкретизируется списком <$b_{k+1} \ldots b_n$>.

3.3 Унификация оператора поиска и заголовка правила

Определим унификацию заголовка оператора поиска *~.xlog_tmpl2.variable2,…* с заголовком правила *~(xlog_tmpl1,variable1,…)*. Заголовок правила и заголовок оператора поиска унифицируются, если унифицируются *xlog_tmpl1* и *xlog_tmp12*, то есть *U(xlog_tmpl1,xlog_tmp12)*.

Переменные входящие в *xlog_tmpl1* или *xlog_tmpl2* после унификации конкретизируются и связываются согласно рассмотренному выше правилу унификации XLog-документов. Переменные заголовка правила и заголовка оператора поиска, не входящие в *xlog_tmpl1* и *xlog_tmp12*, остаются не конкретизированными и несвязанными.

4. Алгоритм исполнения программы

XLog включает следующие механизмы исполнения программы: *унификация, рекурсия и возврат (backtracking).*

Алгоритм исполнения программы аналогичен реализованному в языке Prolog [1] методу логического вывода, основанного на резолюциях.

Исполнение программы начинается с выполнения первой предикатной конструкции из раздела Goal. Семантика всех конструкций XLog рассмотрена, за исключением оператора поиска. Рассмотрим алгоритм исполнения оператора поиска правила.

Оператор поиска последовательно сверху вниз просматривает список правил, пытаясь унифицироваться с их заголовками. Если такое правило найдется, начинается исполнение операторов тела правила. Так же, как в языке Prolog, правило, с которого начинается просмотр, определяет *механизм возврата,* то есть в случае, если оператор в теле правила ложен, то происходит переунификация заголовка правила. Если не находится заголовка, с которым возможна унификация, то оператор поиска считается ложным и происходит возврат (backtracking), к оператору стоящему перед ним.

Поясним алгоритм работы XLog-программы на примере 2.4.1. В этом примере первым исполняется оператор поиска ~.“www.XLog.ru\mysite.html”.

В процессе поиска документ mysite.html унифицируется с шаблоном первого правила (см. пример 1.1). В результате унификации переменные шаблона конкретизируются следующими значениями:

$$X1 – пустая строка,

$$X2 = «Петя послушный мальчик. Петя пошел в школу»,

$$X3 – пустая строка.

Далее последовательно выполняются конструкции тела правила. В нашем случае первым выполняется оператор Split. В результате его исполнения в переменной $$Y1 будет пустой список. Затем исполняется оператор поиска ~.$$Y1.

Оператор ~.$$Y1 унифицируется с правилом 3, состоящим из одного оператора отсечения. Исполнение оператора успешно, поэтому происходит переход к следующему оператору Split, который также успешно исполняется. В результате исполнения второго оператора Split получим список из двух элементов $$Y2 = «Петя послушный мальчик.», «Петя пошел в школу».

Оператор ~.$$Y2 унифицируется с заголовком правила 2, в результате чего переменные заголовка получают следующие значения:

$$X1= «Петя послушный мальчик»,

$$X2= «Петя пошел в школу».

После этого происходит исполнение предикатных конструкций тела правила 2. Первым исполняется оператор поиска ~.<sAnl, $$X1>, который унифицируется с правилом 4.

Затем переходим к исполнению предикатных конструкций правила 4. Сначала выполняется оператор Split, и в результате $$Y= <«Петя», «послушный», «мальчик»>. Затем переходим к выполнению оператора ~.<AtrSnt.$$Y>.

Оператор ~.<AtrSnt. $$Y> не унифицируется с заголовком правила 6, но унифицируется с заголовком правила 7. Все предикатные конструкции правила 7 истинны. Поэтому происходит возврат на конец правила 4.

Далее алгоритм работает аналогичным образом.

5. Интерпретатор

Архитектура программного обеспечения интерпретатора позволяет неограниченно расширять набор точечных операторов. Такое решение оправдано, так как набор точечных операторов во многом определяет эффективность XLog при разработке систем обработки текстовой информации и сильно зависит от поставленной задачи. Точечные операторы, предназначенные для работы с текстовой информацией, часто используют различные лингвистические базы данных, онтологии и т.д. Технология пополнения точечных операторов позволяет эффективно работать с трехуровневой архитектурой доступа к базам данных.

6. Заключение

XLog – новый декларативный язык, предназначенный для работы с HTML/XML-документами. XLog существенно отличается от декларативного языка XSLT. XLog предназначен для анализа и синтеза HTML/XML-документов, в то время как основное назначение XSLT – синтез HTML/XML-документов из XML-документов. Механизмы синтеза в этих языках существо отличаются. В XLog синтез ведется при помощи унификации, в то время как в XSLT для этих целей используются шаблон со специальными тегами. Механизм унификации прост в использовании и позволяет на лету формировать HTML-страницы.

Важной особенность XLog является наличие механизма вывода основанного на правилах и механизме возвратов, который делает язык XLog декларативным. Решение многих задач анализа HTML/XML-документов можно свести к написанию набора правил. Если окажется, что при помощи правил и механизма вывода решать задачу нецелесообразно, то можно воспользоваться включенными в XLog конструкциями из XPath и XQuery.

В интерпретаторе XLog реализован удобный механизм расширения библиотек точечных операторов. Точечные операторы, предназначенные для

обработки текстовой информации, часто используют большие лингвистические базы данных. В программное обеспечение XLog включена технология расширения набора точечных операторов, позволяющая быстро разрабатывать предметно-ориентированные библиотеки. Технология основана на использовании трехуровневой архитектуры, которая позволяет добиваться хороших скоростей обработки информации. Все перечисленное делает XLog эффективным инструментом построения систем интеллектуальной обработки Internet-информации.

Литература

[1] Адаменко А.Н., Кучуков А.М. Логическое программирование и Visual Prolog. – СПб.: БХВ-Петербург, 2003.

[2] Вугт В. Open XML. – Microsoft Press, 2007.

[3] Говард К. W3C XML:XQuery от экспертов. Руководство по языку запросов. – М.: «КУДИЦ-ОБРАЗ», 2005.

[4] D. Chamberlin. XQuery: An XML query language // IBM System Journal, 41:597, 615, 2003.

[5] S. Al-Khalifa et al. Querying structured text in an XML database // In SIGMOD, 2003.

[1] Adamenko A., Kuchukov A. Logic programming and Visual Prolog. – BXB-Petersburg, 2003.

[2] Vugt V. Open XML. – Microsoft Press, 2007.

[3] Howard K. XQuery from the Experts: A Guide to the W3C XML Query Language. – "KUDEC-OBRAZ", 2005.

[4] D. Chamberlin. XQuery: An XML query language // IBM System Journal, 41:597,615, 2003.

[5] S. Al-Khalifa et al. Querying structured text in an XML database. // In SIGMOD, 2003.