

XLog – язык обработки HTML- и XML-документов

© С.С. Магазов, Д.О. Чугунов

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Дано описание основных идей декларативного языка XLog, предназначенного для разработки систем интеллектуальной обработки Internet-информации. Приведены синтаксис и семантика основных конструкций языка, краткое описание архитектуры интерпретатора, а также примеры.

Ключевые слова: декларативный язык, XML/HTML-документ, обработка текстов, Internet, интерпретатор.

Введение. Аналитики Cisco прогнозируют, что к 2015 г. Internet-трафик документов в формате HTML достигнет 245 терабайт в секунду. Постепенно складываются новые реалии в Internet-сообществе и 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 [1, 2].

Язык XPath предназначен для навигации по XML-документам. Язык XQuery, используя конструкции XPath, позволяет организовывать алгоритмы обработки XML-документов [3], а язык XSLT – 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 и структурирование текстовой информации в виде XML/HTML-документа.

XLog предназначен для разработки интеллектуальных систем обработки XLog-документов, которые являются расширением формата XML/HTML, 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 (БД). В статье сформулированы основные идеи языка XLog и дается краткий обзор конструкций языка.

XLog-документ. Множество *XLog-документов* включает 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 видов: *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, т. е. он состоит из открывающего и закрывающего тегов, обрамляющих текст или другие элементы [1].

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 задается следующим образом: $\langle ob_1 \dots ob_n \rangle$ или $\langle ob_1 \dots ob_n | \$\$X \rangle$, где ob_i – либо объект одного из базовых типов, либо список, либо переменная. $\$\X – элементная переменная.

Атрибутивная переменная может стоять на месте значений атрибутов, а элементная – на месте элементов XLog-документа.

Пример 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 результате анализа этого HTML-документа должна быть выделена текстовая информация и проведена 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>

```

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

Точечные операторы.

В реализацию XLog входит библиотека точечных операторов, которая может неограниченно расширяться. Точечные операторы предназначены для обработки текстовой информации. Особенности 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, а также принимать в качестве значения множества элементов XLog-документа. В качестве решения обозначенных выше проблем предлагается конструкция *точечного оператора*, существенно отличающаяся от «классической» конструкции оператора вида $F(x_1 \dots x_n)$. Формат точечного оператора следующий:

$$oper.\{param=var \mid var=param \mid param==var \mid param=const \mid param==const \mid var \mid const\},$$

где *oper* – имя оператора; *param* – имя параметра; *var* – переменная любого 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. В результате исполнения оператор 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 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, формируются значения выходных параметров и проверяются сравнения. Если все сравнения истинны, 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. 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 он возвращает *false*. Операция имеет единственный входной неименованный параметр, указывающий путь.

Пример 3. Пусть переменная $$X$ определяет список элементов $\langle a \rangle$ Element1 $\langle /a \rangle$, $\langle a \rangle$ Element2 $\langle /a \rangle$, тогда выражение $exist. $X \backslash b$ будет ложным, а выражение $exist. $X \backslash *$ – истинным.

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

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

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

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

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

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

Несколько отличный синтаксис у оператора поиска (~). Этот оператор подробно будет рассмотрен далее. Здесь мы определим только его синтаксис. Формат оператора следующий:

$$\sim . xlog_tpl . variable_1, \dots, variable_n,$$

где *xlog_tpl* – это либо URL, либо *xlog*-документ, который задается как *@"xlog"*, либо список; *variable_1, \dots, variable_n* – список (возможно, пустой) переменных, которые могут как входить, так и не входить в *xlog_tpl*.

Оператор навигации по XLog-документу. Любая технология обработки XML/HTML-документов должна включать механизмы доступа к частям такого документа. Язык XPath [5] решает этот вопрос с помощью механизма путей. В XLog включены механизмы из XPath.

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

Путь в XLog имеет следующий вид:

$$path:=(var|URL)(\|\\)ident_1 [[predicate]] \dots (\|\\)ident_n [[predicate]],$$

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.

Результатирующий список на каждом шаге формируется согласно двум правилам:

1) если в текущей позиции пути стоит «\», имя тега (*ident_i*), записанное 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 осуществляется вычисление значения предиката, и элемент добавляется в результирующий список, только если результат равен *true*. Если в квадратных скобках указан индекс возвращаемого элемента, в результирующий список будет помещен элемент с указанным номером (нумерация начинается с 1);

2) если в текущей позиции пути присутствует «\|», выбираются все потомки элемента текущего списка, т. е. списка, полученного на предыдущем шаге, с проверкой предиката на истинность.

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

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 присвоение невозможно, возникает ошибка времени выполнения.

С помощью выражения **in** определяется принадлежность значения переменной *Var* списку элементов, определяемый путем (*path*). Формат выражения следующий:

```
var in path.
```

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

Пример 4. Пусть переменная *\$X* определяет список элементов:

```
<a>Element1</a>,
<a>Element2</a>.
```

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, определяемые путем $path_2$. Переменная var_1 доступна только внутри цикла.

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

Пример 5. Пусть переменная $\$X$ определяет набор элементов:

```
<a><d>Hello1</d></a>;
  <b><b>Hello2</b></b>;
  <a><c>Hello3</c></a>;
  <c><b>Hello4</b></c>.
```

Тогда в результате выполнения выражения

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

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

```
<b>Hello2</b>;
<b>Hello4</b>.
```

Выражение **if** имеет следующий вид:

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

Семантика **if** отличается от общепринятой. Данное выражение вычисляет значение логического выражения. Если результатом является *true*, то вычисляется $predicate_1$, в противном случае – $predicate_2$. Выражение **if** истинно только в том случае, если истинным является предикат, которому передано управление ($predicate_1$ или $predicate_2$). Если выражение **if** истинно, то вычисляется следующая конструкция, в противном случае запускается механизм возврата, рассматриваемый далее.

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

Структура программы. Структура программы на языке XLog похожа на структуру программы на языке PROLOG [4]. Программа состоит из раздела *XLog-правил* (Rule) и раздела *XLog-цели* (Goal). Шаблон программы выглядит следующим образом:

```

Program name
  Rule
    head:- predicate_construction,...;
    ...
    head:- predicate_construction,...;
  head;
  Goal
    predicate_construction, ...;
end

```

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

Заголовок правила имеет три формы:

1) $\sim(xlog_tpl, variable_1, \dots, variable_n)$, где *xlog_tpl* – URL, указывающий на место расположения XLog-документа, а *variable₁, ..., variable_N* – переменные, которые могут как входить, так и не входить в *xlog_tpl*. При этом список переменных может вообще отсутствовать в заголовке.

2) $\sim(@ "xlog", variable_1, \dots, variable_n)$. Эта форма аналогична первой форме. Различие заключается лишь в том, что в первом аргументе задается не URL, а XLog-документ.

3) $\sim(L, variable_1, \dots, variable_n)$, где *L* – список, а *variable_i* – переменные.

Телом правила назовем последовательность предикатных конструкций (*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 6. Приведенная ниже программа выделяет из HTML-документа текстовую информацию. В тексте распознаются упрощенные версии атрибутивных предложений (формула предложения *объект-предикат*) и двухсоставные предложения. *Двухсоставное предложение* содержит два главных члена – подлежащее и сказуемое, причем и подлежащее, 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 двухсоставное предложение";

// Правило 9.

~(<TowPartSnt, <\$\$Y1,\$\$Y2,\$\$Y3,\$\$Y4 >>):-

nWord.str=\$\$Y1.case='nominative', aWord.\$\$Y2,

Preposition.\$\$Y3, nWord.str=\$\$Y4.case='accusative',

print.'двухсоставные предложения';

Goal

~.'www.XLog.ru\mysite.html';

end

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

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

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*, т. е. им присваиваются в качестве значений фрагменты XLog-документа. Если переменной не присвоено значение, она будет *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 $H1 = \langle t1 \ a_1 = V_1 \ \dots \ a_n = V_n \rangle$ и $H2 = \langle t2 \ b_1 = L_1 \ \dots \ b_m = L_m \rangle$, 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 3а–3е.

3а) L и V – константные значения, $L = V$.

3б) L и V – несвязанные и неконкретизированные переменные. В этом случае L и V становятся связанными между собой.

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

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

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

К пунктам 3а–3д добавляются пункты, в которых L и V меняются местами.

4) Унификация XLog-элемента и переменной осуществляется согласно правилам 4а–4б.

4а) если $H1$ – XLog-элемент, а $$$$X$ – неконкретизированная переменная, $H1$ унифицируется с $$$$X$ и $$$$X$ принимает значение $H1$;

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

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

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

5б) Если переменная $\$X$ конкретизирована значением А, а переменная $\$Y$ конкретизирована значением В, тогда унификация успешна в случае, если А и В унифицируются.

5в) Если переменная $\$X$ конкретизирована значением А, а переменная $\$Y$ не конкретизирована, унификация успешна, $\$Y$ примет значение А и станет связана с $\$X$,

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

6) Унификация XLog-элементов выполняется по правилам 6а–6в.

6а) Пусть заданы XLog-элементы

$$\langle t1 \ a_1=V_1 \dots a_n=V_n \rangle H_1 \dots H_p \langle /t1 \rangle ,$$

$$\langle t2 \ b_1=L_1 \dots b_m=L_m \rangle A_1 \dots A_p \langle /t2 \rangle .$$

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

6б) Пусть заданы XLog-элементы

$$A = \langle t1 \ a_1=V_1 \dots a_n=V_n \rangle H_1 \dots H_p | \$X \langle /t1 \rangle ,$$

$$B = \langle t2 \ b_1=L_1 \dots b_m=L_m \rangle A_1 \dots A_k \langle /t2 \rangle$$

и $p \leq k$, тогда А и В унифицируются, если заголовок $\langle t1 \ a_1=V_1 \dots a_n=V_n \rangle$ унифицируется с $\langle t2 \ b_1=L_1 \dots b_m=L_m \rangle$, а также $H_1 \dots H_p$ и $A_1 \dots A_p$ попарно унифицируются. Причем если $\$X$ не конкретизирована, она принимает значение $A_{p+1} \dots A_k$, если же $\$X$ конкретизирована, значение должно унифицироваться с $A_{p+1} \dots A_k$.

6в) Пусть заданы XLog-элементы

$$A = \langle t1 \ a_1=V_1 \dots a_n=V_n \rangle H_1 \dots H_p | \$X \langle /t1 \rangle ,$$

$$B = \langle t2 \ b_1=L_1 \dots b_m=L_m \rangle A_1 \dots A_p | \$Y \langle /t2 \rangle .$$

Тогда А и В унифицируются, если унифицируются заголовки, $H_1 \dots H_p$ и $A_1 \dots A_p$ попарно унифицируются, а $\$X$ унифицируется с $\$Y$.

Пример 7. В результате унификации HTML-документа и шаблона из примера 1 получим: $\$X1$ связана с пустой строкой, $\$X2$ – со значением «Петя послушный мальчик. Петя пошел в школу», переменная $\$X3$ – с пустой строкой.

Унификация списков. Списки $\langle b_1 \dots b_n \rangle$ и $\langle c_1 \dots c_n \rangle$ унифицируются, если $U(b_i, c_i)$ для всех $i \leq n$.

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

Унификация оператора поиска и заголовка правила. Определим унификацию заголовка оператора поиска $\sim xlog_tpl2.variable_2, \dots$ с заголовком правила $\sim(xlog_tpl1, variable_1, \dots)$. Заголовок правила и заголовок оператора поиска унифицируются, если унифицируются $xlog_tpl1$ и $xlog_tpl2$, т. е. $U(xlog_tpl1, xlog_tpl2)$.

Переменные, входящие в $xlog_tpl1$ или $xlog_tpl2$, после унификации конкретизируются и связываются согласно рассмотренному выше правилу унификации XLog-документов. Переменные заголовка правила и заголовка оператора поиска, не входящие в $xlog_tpl1$ и $xlog_tpl2$, остаются неконкретизированными и несвязанными.

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

Алгоритм исполнения программы аналогичен реализованному в языке PROLOG [4] методу логического вывода, основанного на резолюциях.

Реализация 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*, т. е. в случае, если оператор в теле правила ложен, происходит переунификация заголовка правила. Если не находится заголовка, с которым возможна унификация, оператор поиска считается ложным и происходит возврат (backtracking) к оператору, стоящему перед ним.

Поясним алгоритм работы XLog-программы на примере 6. В этом примере первым исполняется оператор поиска \sim . “www.XLog.ru\mysite.html”.

В процессе поиска документ mysite.html унифицируется с шаблоном первого правила (см. пример 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 выполняется. 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 выполняется оператор поиска ~.<sAnl, \$\$X1>, который унифицируется с правилом 4.

Затем переходим к исполнению предикатных конструкций правила 4. Сначала выполняется оператор Split, 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 в конец правила 4.

Далее алгоритм работает аналогичным образом.

Интерпретатор. Архитектура программного обеспечения интерпретатора позволяет неограниченно расширять набор точечных операторов. Это оправданно, 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 БД, онтологии и т. д. Технология пополнения точечных операторов позволяет эффективно работать с трехуровневой архитектурой доступа к БД.

Заключение. XLog – новый декларативный язык для работы с HTML/XML-документами. XLog существенно отличается от декларативного языка XSLT и 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 значительно отличаются: в XLog синтез ведется с помощью унификации, а в XSLT для этих целей используется

шаблон со специальными тегами. Механизм унификации прост в работе и позволяет на лету формировать HTML-страницы.

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

В интерпретаторе XLog реализован удобный механизм расширения библиотек точечных операторов, предназначенных для обработки текстовой информации. Они часто используют большие лингвистические базы данных. В программное обеспечение XLog включена технология расширения набора точечных операторов, что дает возможность 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] D. Chamberlin. *XQuery: An XML query language*. *IBM System Journal*, 2003, 41:597, p. 615.
- [2] Al-Khalifa S. et al. *Querying structured text in an XML database*. N.Y., SIGMOD, 2003.
- [3] Вугт В. *Open XML*. Microsoft Press, 2007.
- [4] Адаменко А.Н., Кучуков А.М. *Логическое программирование и Visual Prolog*. Санкт-Петербург, БХВ-Петербург, 2003.
- [5] Говард К. *W3C XML:XQuery от экспертов: руководство по языку запросов*. Москва, КУДИЦ-ОБРАЗ, 2005.

Статья поступила в редакцию 24.06.2013

Ссылку на эту статью просим оформлять следующим образом:

Магазов С.С., Чугунов Д.О. XLog – язык обработки HTML- и XML-документов. *Инженерный журнал: наука и инновации*, 2013, вып. 11 URL: <http://engjournal.ru/catalog/it/hidden/1061.html>

Магазов Сергей Салимович родился в 1954 г., окончил Казанский государственный университет в 1976 г. Канд. физ.-мат. наук, доцент кафедры «Теоретическая информатика и компьютерные технологии» МГТУ им. Н.Э. Баумана. Автор 40 научных работ и трех монографий. Специализируется в области цифровой обработки сигналов и методов искусственного интеллекта. e-mail: 2006mag@mail.ru

Чугунов Дмитрий Олегович – студент кафедры «Теоретическая информатика и компьютерные технологии» МГТУ им. Н.Э. Баумана. Специализируется в областях конструирования компиляторов и баз данных NoSQL. e-mail: d.chugunov11@mail.ru