Dagstuhl-Seminar on *Rule Markup Techniques*

7th of February, 2002

XL: A rule-based query and transformation language for XML and SSD

François Bry and Sebastian Schaffert
Ludwig-Maximilians-Universität München
http://www.pms.informatik.uni-muenchen.de

1. Motivation

2. XML and Terms

3. Elements of a Query and Transformation Language

   - Construct-Query Rules
   - Database Terms
   - Query Terms
   - Construct Terms

4. Simulation Unification

5. Conclusion

Imagine two online bookstores that provide a list of books with, among other things, their titles and prices. Bookstore A:

Example

```
<bib>
  <book>
    <title>Cryptonomicon</title>
    <authors>
      <author>Alice</author>
      <author>Bob</author>
    </authors>
    <price>39.95</price>
  </book>
  <book>
    <title>Applied Cryptography</title>
    <author>Alice</author>
    <price>34.95</price>
  </book>
  ...
</bib>
```

Bookstore B, on the other hand could provide a list in the style of the following excerpt:

Example

```
<reviews>
  <entry>
    <title>Applied Cryptography</title>
    <price>36.95</price>
    <comment>A good book on cryptography</comment>
  </entry>
  <entry>
    <title>Cryptonomicon</title>
    <price>31.95</price>
    <comment>A must-have for your private intelligence service</comment>
  </entry>
  ...
</reviews>
```

A common query for such heterogenous sources could be:

Give me a list of all books with a comparison of its price at store A and B

The result for the example databases would look as follows:

Example

```
<books-with-prices>
  <book-with-prices>
    <title>Applied Cryptography</title>
    <price-A>34.95</price-A>
    <price-B>36.95</price-B>
  </book-with-prices>
  <book-with-prices>
    <title>Cryptonomicon</title>
    <price-A>39.95</price-A>
    <price-B>31.95</price-B>
  </book-with-prices>
  ...
</books-with-prices>
```

In a "navigational" query language like the XPath-based *XQuery* [2] and *XSLT* [1] this query would consist of **several independant "subqueries"** for each of the databases:

1. find all entries in the database that have a title and a price (`/bib/book[title and price]`)

2. for each of the entries, retrieve the title (`./title`)

3. for each of the entries, retrieve the price (`./price`)

It is easy to observe that there is no (immediate) connection between these subqueries other than the sequence in which they are evaluated.

Furthermore, the **construction part** and the **query part** are **tightly integrated**.

In XQuery, the query would look like this:

Example

```
<books-with-prices>
   { FOR $a in document("A/bib.xml")//book,
         $b in document("B/reviews.xml")//entry
     WHERE $b/title = $a/title
     RETURN
        <book-with-prices>
           { $b/title }
           <price-A>
             { $a/price/text() }
           </price-A>
           <price-B>
             { $b/price/text() }
           </price-B>
        </book-with-prices>
    }
</books-with-prices>
```

In contrast to such a navigational way of querying, we propose a rule-based approach – similar to languages like Prolog – with the following two main features:

- term-based ("positional") querying with a *template* of the data in the database

- rule-based programs with a clear separation between construction- and query part

It is our conviction that the declarativeness of such a language . . .

- will make it easier to use in many cases (it may even be possible to create a visual interface for it)

- will make complex transformations more obvious (and thus lead to easier maintainability)

In our XL-approach, the query could look like this:

Example

```
construct
   <book-with-prices>
      <title>T</title>
      <price-A>Pa</price-A>
      <price-B>Pb</price-B>
   </book-with-prices>
where
   in A/bib.xml:
   <book>
      <title>T</title>
      <price>Pa</price>
   </book>
   and
   in B/reviews.xml:
   <entry>
      <title>T<title>
      <price>Pb</price>
   </entry>
```

Term representations of XML data are straightforward:

Example

```
<bib>
  <book>
    <title>Cryptonomicon</title>
    <authors>
      <author>Alice</author>
      <author>Bob</author>
    </authors>
    <price>39.95</price>
  </book>
  <book>
    <title>Applied Cryptography</title>
    <author>Alice</author>
    <price>34.95</price>
  </book>
  ...
</bib>
```

Example

```
bib(
  book(
    title('Cryptonomicon'),
    authors(
      author('Alice'),
      author('Bob')
    ),
    price(39.95)
  ),
  book(
    title('Applied Cryptography'),
    author('Alice'),
    price(34.95)
  ),
  ...
)
```

However, it is not so easy to apply the methods of logic programming to such terms:

1. XML is *semi*-structured:
   - structure may be incomplete
   - a given structure (DTD, XML Schema) may be ignored
   - several entries of similar kind may have differing structure
2. Data is organized in a different way than in traditional term-based approaches:
   - alternatives are nested within the same term instead of using several terms
   - order may or may not be of relevance, depending on the application

A term-based language has to cope with these properties. In the *XL*-project, we propose:

- a *term language* that provides constructs for dealing with unknown and flexible structure
- a non-standard *unification algorithm* that makes use of these constructs for querying flexible data with nested alternatives
- a *rule* language building on top of the two latter concepts

Construct-Query Rules

A program in the language XL consists of one or more rules of the style

$$\underset{\text{Head}}{t^c} \quad \leftarrow \quad \underset{\text{Body}}{t_1^q \wedge \cdots \wedge t_n^q}$$

where each term in the *body* is evaluated against a (possibly different) database or head of another rule. The *head* is used to "construct" the answer.

Both backward and forward chaining of rules is possible in the current approach.

Database Terms

Database Terms are an abstraction of XML documents.

- $l[t_1, \ldots, t_n]$ is a database term with the root labelled l and the sequence of children $t_1, \ldots, t_n$ is *ordered*

- $l\{t_1, \ldots, t_n\}$ is a database term with the root labelled l and the bag of children $t_1, \ldots, t_n$ is *unordered*

Instead of $l[]$ and $l\{\}$ (i.e. $n = 0$), we write simply l.

Query Terms

Query Terms . . .

- are a *pattern* for the data in the database
- contain variables in order to retrieve information from the database

Query Terms

In contrast to Prolog goals, however, Query Terms have the following additional properties:

- subterms with additonal structure might be answers
- subterms with different subterm ordering might be answers
- the query term might specify subterms at an unspecified depth

Query Terms

In our abstract syntax, we write Query Terms similarly to Database Terms, with the following additional properties:

- double parentheses ([[]] and $\{\{\}\}$) are used to specify a *total* matching, while single parentheses express *partial* matching
- the *descendant* construct allows to represent subterms at an unspecified depth ($desc\ t$)
- *variables* refer to subterms in the Query Term ($X \rightsquigarrow t$, read X "as" t)

Obviously, such a flexible structure implies that there might be several alternative answers for a query.

Query Terms – Example

$$bib\{book\{T \rightsquigarrow title, desc\ author\{A \rightsquigarrow \cdot\}\}\}.$$

might be an appropriate query term for a structure where the depth of the `author` elements below a `book` is not known:

Example

```
<bib>
  <book>
    <title>Applied Cryptography</title>
    <author>Alice</author>
  </book>
  <book>
    <title>Cryptonomicon</title>
    <authors>
      <author>Alice</author>
      <author>Bob</author>
    </authors>
  </book>
</bib>
```

Construct Terms

Construct Terms serve to reassemble variables in a new structure:

- *variables* only occur in plain form (i.e. no $\rightsquigarrow$)
- *term quantifiers* may be used to iterate over possible alternative answers that result from the evaluation of the query part
- are otherwise similar to Database Terms

$$all\ author[name[A], titles\{all\ T\}] \leftarrow bib\{book\{T \rightsquigarrow title, desc\ author\{A \rightsquigarrow \cdot\}\}\}.$$

would yield the following result:

Example

```
<author>
  <name>Alice</name>
  <titles>
    <title>Applied Cryptography</title>
    <title>Cryptonomicon</title>
  </titles>
</author>
<author>
  <name>Bob</name>
  <titles>
    <title>Cryptonomicon</title>
  </titles>
</author>
```

Instead of using standard unification, we propose a non-standard *Simulation Unification* that is more suited for the properties of XML data:

- Our *Simulation Unification* algorithm works on a formula with conjunctions and disjunctions of *inequalities* between Query Terms and Construct Terms.

- Such inequalities are the result of a relation called *simulation* which we use to define a *lattice* (i.e. partial ordering with $\top$ and $\bot$) over the set of terms

- The algorithm consists of two phases, a *Term Decomposition* and a *Consistency Verification* for variables

Simulation

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs and let $\sim$ be an equivalence relation on $V_1 \cup V_2$. A relation $\mathcal{S} \subseteq V_1 \times V_2$ is a simulation (with respect to $\sim$) if:

1. If $(v_1, v_2) \in \mathcal{S}$, then $v_1 \sim v_2$.

2. If $(v_1, v_2) \in \mathcal{S}$ and $(v_1, v_1') \in E_1$, then there exists $v_2' \in V_2$ such that $(v_1', v_2') \in \mathcal{S}$ and $(v_2, v_2') \in E_2$

A simulation on two *trees* is called *rooted*, if the root nodes of the two trees are part of the simulation.

Given two terms t_1, t_2. Let $\preceq$ be the preorder defined by $t_1 \preceq t_2$ if there exists a (rooted) simulation from t_1 to t_2.

Simulation – Example

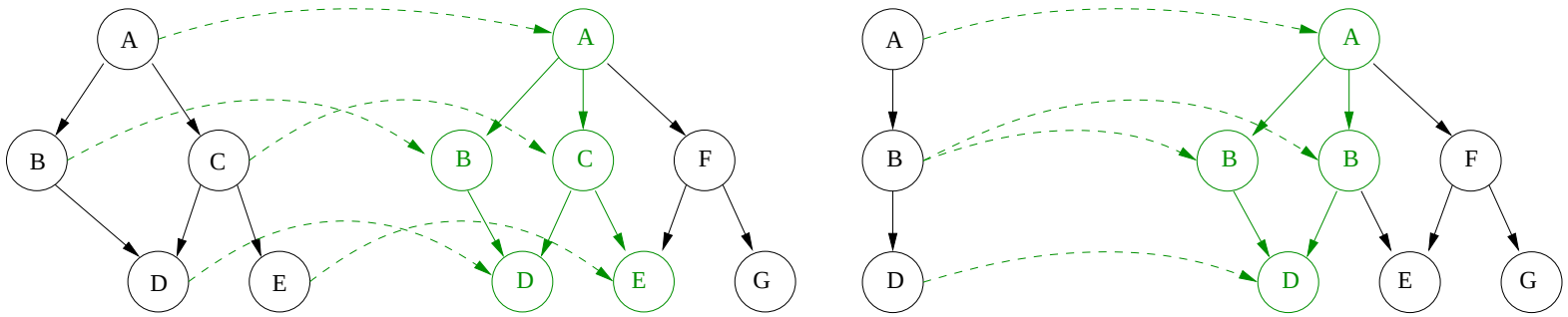

Figure 1: Simulations (with respect to label equality)

Deduction

XL-programs can be evaluated using

- *backward reasoning* as in programming languages like Prolog
- *forward reasoning* as in deductive databases

The evaluation approach is similar to *constraint solving over finite domains*:

- variables are not bound until the end of the deduction process
- the evaluation works on the upper/lower bounds of variables
- in the deduction process, the interval on a variable may be reduced

Deduction – Example

Given the following query:

$$\leftarrow q(X \rightsquigarrow a), r(X \rightsquigarrow a).$$
$$q(a(b,c)).$$
$$r(a(b,d)).$$

When evaluating $q(X \rightsquigarrow a)$, we do not immediately bind the variable X. Instead, we generate the inequality:

$$a \preceq X \preceq a(b,c)$$

after also evaluating $r(X \rightsquigarrow a)$, an additional constraint is added:

$$a \preceq X \preceq a(b,d)$$

The two constraint are then simplified to

$$a \preceq X \preceq a(b)$$

Algorithm – Decomposition 1

- **Root Elimination:**

(1) $l \preceq l\{t_1^2, \ldots, t_m^2\} \Leftrightarrow \mathit{true}$ if $m \geq 0$

(2) $l\{t_1^1, \ldots, t_n^1\} \preceq l \Leftrightarrow \mathit{false}$ if $n \geq 1$

(3) Let Π be the set of (total) functions $\{t_1^1, \ldots, t_n^1\} \rightarrow \{t_1^2, \ldots, t_m^2\}$:
$l\{t_1^1, \ldots, t_n^1\} \preceq l\{t_1^2, \ldots, t_m^2\} \Leftrightarrow \bigvee_{\pi \in \Pi} \bigwedge_{1 \leq i \leq n} t_i^1 \preceq \pi(t_i^1)$
if $n \geq 1$ and $m \geq 1$

(4) $l_1\{t_1^1, \ldots, t_n^1\} \preceq l_2\{t_1^2, \ldots, t_m^2\} \Leftrightarrow \mathit{false}$ if $l_1 \neq l_2$ and $n \geq 0$ and $m \geq 0$

Algorithm – Decomposition 2

- $\rightsquigarrow$ **Elimination:**

$$X \rightsquigarrow t^1 \preceq t^2 \quad \Leftrightarrow \quad t^1 \preceq t^2 \ \wedge \ t^1 \preceq X \ \wedge \ X \preceq t^2$$

- **Descendant Elimination:**

$$desc\ t^1 \preceq l_2\{t_1^2, \ldots, t_m^2\} \Leftrightarrow t^1 \preceq l_2\{t_1^2, \ldots, t_m^2\} \vee \bigvee_{1 \leq i \leq m} desc\ t^1 \preceq t_i^2$$

if $m \geq 0$

Algorithm – Consistency Verification

- **GLB/LUB Merge:**

(1)	$X \preceq t_1^{db} \wedge X \preceq t_2^{db} \Leftrightarrow \mathit{false}$	if $glb(t_1^{db}, t_2^{db}) = \bot$
(2)	$X \preceq t_1^{db} \wedge X \preceq t_2^{db} \Leftrightarrow X \preceq glb(t_1^{db}, t_2^{db})$	if $glb(t_1^{db}, t_2^{db}) \neq \bot$
(3)	$t_1^{db} \preceq X \wedge t_2^{db} \preceq X \Leftrightarrow \mathit{false}$	if $lub(t_1^{db}, t_2^{db}) = \top$
(4)	$t_1^{db} \preceq X \wedge t_2^{db} \preceq X \Leftrightarrow lub(t_1^{db}, t_2^{db}) \preceq X$	if $lub(t_1^{db}, t_2^{db}) \neq \top$

- **Transitive Closure:**

$$t_1 \preceq X \wedge X \preceq t_2 \Rightarrow t_1 \preceq t_2$$

Algorithm – Example

$$\{\ f[Y \rightsquigarrow a\{b\}, g\{Y \rightsquigarrow a\{c\}\}]) \preceq f[a[b], a[b,c], g[a[b,c,d]]]\ \} \qquad \longrightarrow_{\text{Root Elim. (3)}}$$

$$\begin{aligned}
\{\ &(\ Y \rightsquigarrow a\{b\} \preceq a[b] \wedge g\{Y \rightsquigarrow a\{c\}\} \preceq a[b,c]\ ) \vee \\
&(\ Y \rightsquigarrow a\{b\} \preceq a[b] \wedge g\{Y \rightsquigarrow a\{c\}\} \preceq g[a[b,c,d]]\ ) \vee \\
&(\ Y \rightsquigarrow a\{b\} \preceq a[b,c] \wedge g\{Y \rightsquigarrow a\{c\}\} \preceq g[a[b,c,d]]\ )\ \}
\end{aligned} \qquad \longrightarrow^{*}_{\text{Root Elim. (3)}}$$

$$\begin{aligned}
\{\ &(\ Y \rightsquigarrow a\{b\} \preceq a[b] \wedge Y \rightsquigarrow a\{c\} \preceq a[b,c,d]\ ) \vee \\
&(\ Y \rightsquigarrow a\{b\} \preceq a[b,c] \wedge Y \rightsquigarrow a\{c\} \preceq a[b,c,d]\ )\ \}
\end{aligned} \qquad \longrightarrow^{*}_{\rightsquigarrow-\text{Elim.}}$$

$$\begin{aligned}
\{\ &(\ a\{b\} \preceq Y \wedge Y \preceq a[b] \wedge a\{b\} \preceq a[b] \wedge \\
&\quad a\{c\} \preceq Y \wedge Y \preceq a[b,c,d] \wedge a\{c\} \preceq a[b,c,d]\ ) \vee \\
&(\ a\{b\} \preceq Y \wedge Y \preceq a[b,c] \wedge a\{b\} \preceq a[b,c] \wedge \\
&\quad a\{c\} \preceq Y \wedge Y \preceq a[b,c,d] \wedge a\{c\} \preceq a[b,c,d]\ )\ \}
\end{aligned} \qquad \longrightarrow^{*}_{\text{Root Elim.}}$$

Algorithm – Example cont.

$$\{ \; (\; a\{b\} \preceq Y \wedge Y \preceq a[b] \wedge a\{c\} \preceq Y \wedge Y \preceq a[b,c,d] \;) \vee$$

$$(\; a\{b\} \preceq Y \wedge Y \preceq a[b,c] \wedge a\{c\} \preceq Y \wedge Y \preceq a[b,c,d] \;) \; \} \qquad \longrightarrow^{*}_{\text{GLB/LUB}}$$

$$\{ \; (\; a\{b,c\} \preceq Y \wedge Y \preceq a[b] \;) \vee (\; a\{b,c\} \preceq Y \wedge Y \preceq a[b,c] \;) \; \} \qquad \longrightarrow^{*}_{\text{Transitivity}}$$

$$\{ \; (\; a\{b,c\} \preceq Y \wedge Y \preceq a[b] \wedge a\{b,c\} \preceq a[b] \;) \vee$$

$$(\; a\{b,c\} \preceq Y \wedge Y \preceq a[b,c] \wedge a\{b,c\} \preceq a[b,c] \;) \; \} \qquad \longrightarrow^{*}_{\text{Root Elim.}}$$

$$\{ \; (\; a\{b,c\} \preceq Y \wedge Y \preceq a[b] \wedge false \;) \vee$$

$$(\; a\{b,c\} \preceq Y \wedge Y \preceq a[b,c] \wedge true \;) \; \} \qquad \longrightarrow$$

$$\{ \; a\{b,c\} \preceq Y \wedge Y \preceq a[b,c] \; \}$$

Perspectives

- our rule language may be used as the base for *visual querying*
- *schema validation* ("type checking") can be expressed in a very similar way
- schema information provides an interesting way for optimizing the queries (i.e. reducing the alternatives)
- an XL-program is just an "implementation" of an ontology so automatic "mining" of rules from an ontology might by possible

Summary

- declarative, term-based querying may have advantages over the current navigational approaches
- term languages have a long tradition in logic programming, but the underlying assumptions are to strict to deal with semi-structured data

We propose a language called *XL* that tries to address these convictions. XL currently consists of . . .

- a preliminary rule-based language for querying and transforming XML and SSD
- a corresponding evaluation strategy based on a non-standard unification algorithm

References

[1] W3C, http://www.w3.org/Style/XSL/. *Extensible Stylesheet Language (XSL)*, 2000.

[2] W3C, http://www.w3.org/TR/xquery/. *XQuery: A Query Language for XML*, Feb 2001.