XSL Transformations,
Database Queries, and
Computation

Pekka Kilpeläinen
University of Helsinki
Department of Computer Science

February 18, 2000

1. Introduction and Overview

XSLT is a recent special-purpose language for transforming XML documents

Expressive power of XSLT?

Usability of XSLT for typical data manipulation?

Results

- Ways to simulate relational algebra using XSLT operations
- Any computation can be simulated using XSLT

2. XSLT (XSL Transformations)

W3C Recommendation, Nov. 16, 1999

a language for transforming XML documents into other XML documents

XML (Extensible Markup Language).
W3C Recommendation, 1998

a way of representing hierarchical structures (trees) by marked-up documents

Example: See: <fig ref="pic1.gif"/>

No (processing, formatting, or rendering) semantics fixed for the markup; structure and content the only concerns

~ formatting semantics attached to XML documents using stylesheets

Primary purpose of XSLT: to be used together with the (forthcoming) XML stylesheets language XSL

Overview of the rest of the talk

2. XSL transformations
3. Relational database model
4. Relational queries using XSLT
5. XSLT computability
6. Conclusion
XSLT uses XPath (XML Path Language). W3C Rec., Nov. 16, 1999, for addressing parts of XML documents.

Also in XPath: basic manipulation of strings, numbers and booleans.

Major components of XML trees

- **Element nodes** as internal nodes
- **Attribute nodes** attached to element nodes
- **Text nodes** (or empty element nodes) as leaves

XPath & XSLT: an additional root node, `'/`.

(Cf. Unix/Linux file system hierarchy)

---

XSLT syntax and semantics

An XSLT stylesheet is an XML document that creates a **result tree** by applying **templates** to nodes of the **source tree**.

A mixture of XSLT instructions and XML markup for result tree fragments.

Uses XML namespaces to distinguish between XSLT instructions and result fragments.

Separation of (possibly overlapping) sets of names by using a local prefix (e.g. `xsl:`) bound to a globally unique URI (e.g., `http://www.w3.org/1999/XSL/Transform`).

---

XPath tree for the Example document

```
/ Example
 /     'See:'
 /  'fig'
 /     ref-pic1.gif
```

---

An example

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/Example">
    <html><head></head><body>
      <xsl:apply-templates/>
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

Result

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
  <html>
    <head></head>
    <body>
      See: <img src="pic1.gif" alt="Some picture"/>
    </body>
  </html>
```
3. Relational database model
(Codd 1970)

Basis of today's commercial relational databases

database: collection of uniquely named tables (or relations)
relation is a set of rows (or tuples)
column = attribute

Relational algebra
a procedural query language for basic manipulation of tabular data

Fundamental operations
1. select, $\sigma_{\theta}(R)$, where $\theta$ is a condition on rows of relation $R$
2. project, $\pi_A(R)$, where $A$ is list of attributes of $R$
3. Cartesian product, $R_1 \times R_2$
4. union, $R_1 \cup R_2$
5. set difference, $R_1 - R_2$
6. renaming of attributes, $\rho_{A\to B}(R)$

Others based on the above operations, e.g., $\text{join } r \times s \equiv \pi \ldots \sigma \ldots (r \times s)$

4. Relational queries using XSLT
What's the point?
XSLT is not a DB implementation language!

Answer:
1. Study of theoretical expressiveness
2. Study of practical usability (or of XSLT programming techniques) in typical data manipulation
Database as an XML document

```xml
<db descr="A toy credit card database">
    <acc id="1" name="Smith" prov="Ont" cha=150.15 hrex=3000/>
    <acc id="2" name="Jones" prov="BC" cha=2014.00 hrex=2900/>
    <acc id="3" name="Doc" prov="Ont" cha=190.00 hrex=4000/>
</acc>
</db>
```

How to select, say ‘Vendors in Waterloo’?

**Relational algebra:** π<sub>City=‘Waterloo’</sub>(V)

**XSLT** (using a literal result element):

```xml
<db xml:space="preserve">
    <result descr="Rows of V for vendors in Waterloo">$
    <xsl:for-each select="/V[City='Waterloo']">
        <vendor vno="${vno}" vname="${vname}" city="${city}" vbal="${vbal}"/>
    </xsl:for-each>
    </result>
</db>
```

Result:

```xml
<vendor vno="2" vname="Kmart" city="Waterloo" vbal="671.05"/>
<vendor vno="4" vname="Esso" city="Waterloo" vbal="2.26"/>
```

How to project, say, names and cities of vendors?

**Relational algebra:** π<sub>Vname,City</sub>(V)

**XSLT:**

```xml
<db xml:space="preserve">
    <result descr="Vendor projected on Vname and City">$
    <xsl:for-each select="/V">
        <vendor vname="${vname}" city="${city}"/>
    </xsl:for-each>
    </result>
</db>
```

Result (slightly modified):

```xml
<vendor vname="Sears" city="Toronto" vbal="200.00"/>
<vendor vname="Kmart" city="Waterloo" vbal="671.05"/>
<vendor vname="Esso" city="Waterloo" vbal="2.26"/>
```

How to do Cartesian product, R × S?

**Sample DB as an XML document**

```xml
<db descr="A formal database example">
    <r t="1" b="x"/>
    <r t="2" b="y"/>
    <s t="a" d="a"/>
    <s t="b" d="b"/>
    <s t="c" d="c"/>
</db>
```

**Solution principle**

- apply two nested `xsl:for-each` loops
- store the attributes of the outer-loop tuple in an `xsl:variable`
XSLT transformation for $R \times S$

```xml
<db xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:variable name="RAttributes" select="@RAtttrs"/>

<result descr="Vendor relation with Vbal renamed to Balance">
<xs:if test="@Vname = 'Toronto'">
<xsl:copy-of select="@Vbal"/>
</xs:if>
</result>
</db>
```

How to do union, $R \cup S$?

"Vendor#’s for vendors either in Waterloo or having some transactions?"

$$\pi_{Vno \circ City}(V) \cup \pi_{Vno}(T)$$

XSLT can simply pick rows of each relation in turn:

```xml
<db xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<result descr="Vendor numbers for which there are no transactions">
<xs:for-each select="/V/t/">
<xsl:variable name="Vname" select="$Vno"/>

<xs:for-each select="/@City">
<xsl:variable name="City" select="$City"/>
</xs:for-each>
</result>
</db>
```

Tuples in the result

```
<xs:for-each select="/V/t/">
<xs:for-each select="/@City">
<xs:copy-of select="$Vname"/>
</xs:for-each>
</xs:for-each>
```

Elimination of duplicates also possible (but slightly more complicated)

```
<xs:for-each select="/V/t/">
<xs:for-each select="/@City">
<xs:copy-of select="$Vname"/>
</xs:for-each>
</xs:for-each>
```

How to do set difference, $R - S$?

"Vendor numbers for which there are no transactions?"

$$(\pi_{Vno}(V)) - (\pi_{Vno}(T))$$

XSLT code for set difference

```xml
<db xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<result descr="Vendor numbers without transactions">
<xs:for-each select="/V/t/">
<xsl:variable name="Vname" select="$Vno"/>

<xs:for-each select="/@City">
<xsl:variable name="City" select="$City"/>
</xs:for-each>
</result>
</db>
```

How to rename attributes, $\rho_A \rightarrow p(R)$?

"Rename attribute Vbal in relation V to Balance"

XSLT can supply a new name for an `xsl:attribute`:

```xml
<db xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<result descr="Vendor relation with Vbal renamed to Balance">
<xs:for-each select="/V/t/">
<xsl:attribute name="Balance"/>
</xs:for-each>
</result>
</db>
```

Tuples in result

```
<xs:for-each select="/V/t/">
<xsl:attribute name="Balance"/>
</xs:for-each>
```

```
<xs:for-each select="/V/t/">
<xsl:attribute name="Balance"/>
</xs:for-each>
```
Discussion

Result tree fragments can be processed only once

... a relational query must be simulated in a single transformation step

(Or using a sequence of XSLT processes, each working on the result of the previous one)

Implementations offer extensions for processing node sets multiple times; built-in support expected in further versions of XSLT

Additional relational operations also possible:

- **Sorting** supported by `xsl:sort`
- **Grouping** possible with XSLT programming tricks; built-in support expected in further versions of XSLT

5. XSLT Computability

XSLT can simulate any computation

Shown by simulating Turing machines

**Turing machine** (Turing 1936/7)

- A formal model of algorithms
- Primitive but powerful: any computation expressible in any algorithmic model can be simulated (Church/Turing thesis)

- A potentially infinite tape of cells for symbols, examined by a tape head

- A finite set of states

Control of a Turing machine

Simplistic control defined by a *transition function* \( \sigma \)

Given the current state \( q_1 \) and the symbol at the tape head \( a \)

\[ \sigma(q_1, a) = (q_2, b, X) \]

specifies the next state \( q_2 \), symbol \( b \) written in the cell pointed by head, and one-step movement \( X \in \{ L, R \} \) of the head (Left/Right)

Programming-like features of XSLT

Basic string manipulation capabilities of XPath: substrings, concatenation

- Conditional processing `xsl:choose`, `xsl:when`

  (Cf. *case*)

- Template rules can be named and called with parameters recursively

Simulating a TM by an `xsl:template`

Recursive template rule named `"transition"`

**Parameters** of `"transition"`:

- state: name of the current state
- left: contents of the tape up to the head
- right: tape from the head onwards
Overall structure of the simulation

```xml
<xsl:template name="transition"/>

<xsl:choose>
  <xsl:when test="@state = 'YES'">
    <xsl:value-of select="ACCEPT"/>
  </xsl:when>

  <!-- For each transition defined for @state and the first char of @right, update @state, @left and @right, and call "transition" recursively -->

  <xsl:otherwise>
    <xsl:value-of select="Decl"/>
  </xsl:otherwise>
</xsl:choose>

Updating the representation of the tape

For right-moves \( \sigma(q_1, a) = (q_2, b, R) \)
concatenate 'b' at the end of @left and drop the first char of @right

Left-moves \( \sigma(q_1, a) = (q_2, b, L) \) in a similar manner

Sample trace of the simulation

```xml
<xm dummy.xml tm_palindrome.xsl input_tape=abba
  <xsl:when test="@state = 'YES'">
    <xsl:value-of select="ACCEPT"/>
  </xsl:when>

  <!-- For each transition defined for @state and the first char of @right, update @state, @left and @right, and call "transition" recursively -->

  <xsl:otherwise>
    <xsl:value-of select="Decl"/>
  </xsl:otherwise>
</xsl:choose>

Simulation of a single transition:

```xml
<xml:when test="@state = 'YES' and substring(@right, 1, 1) = '1'">

  <!-- For each transition defined for @state and the first char of @right, update @state, @left and @right, and call "transition" recursively -->

  <xml:value-of select="Decl"/>
</xml:when>

Implications

XSLT has, in principle, unrestricted computational power

~ Not possible to set any limit on the complexity of XSL stylesheet evaluation

~ Not possible to automatically recognize non-terminating stylesheets

~ A malicious programmer could create stylesheets that jam client applications
6. Conclusion

**XSLT**
- Language for transforming XML documents
- Relationally complete (query) language
- Turing complete (programming) language

**Related work**

Gonnet, Tompa & Blakeley (1987): Simulation of relational algebra using *p-string algebra*

Tärnlund (1977): Horn clause computability

**Further work**

Relation of XSLT

with other transformation formalisms?

with proposed XML query languages?