Tree Query and Transformation Techniques

(part 2)

Gabor Guta

Research Institute for Symbolic Computation

23th May 2007
Content

- Standard Query and Transformation approaches
  - Overview of XPath expressions
  - Basic elements of XSLT
  - Introduction to XQuery
- Research approaches
  - XDuce
- Theory of transformation
  - Macro-tree
- Type-checking problem of the transformation
- Goals of my PhD
Standard Query and Transformation approaches

- Programming API-s for existing languages
  - SAX - event based
  - DOM - object based
  - streaming API-s (pull model)

- dedicated languages
  - XSLT
  - XQuery
  - additional standards: XPath, XPointer, XSL-FO, etc.

- usages in database technologies (SQL extensions)
Example XML

The examples are based on the XMP Q1 from the XQuery Use Cases - W3C Working Draft 12 November 2003
The DTD of the Example

<!ELEMENT bib (book* )>
<!ELEMENT book (title, (author+ | editor+ ),
   publisher)>
<!ATTLIST book year CDATA  #REQUIRED >
<!ELEMENT author (last, first )>
<!ELEMENT editor (last, first, affiliation )>
<!ELEMENT title (#PCDATA )>
<!ELEMENT last (#PCDATA )>
<!ELEMENT first (#PCDATA )>
<!ELEMENT affiliation (#PCDATA )>
<!ELEMENT publisher (#PCDATA )>
Overview of XPath expressions

- Navigation expression in a tree
- Concept of axis: child, parent, etc.
XPath examples

- selects the root node: /bib
  or /child::bib/child::book
- selects all nodes of the context node: *
XSLT

- XSLT describes rules for transforming a source tree into a result tree*
- Elements from the source tree can be filtered and reordered, and arbitrary structure can be added*
- The most widely accepted standard
- Mixture of rule based and imperative approach
- XSLT described as an XML!

* XSLT - W3C Recommendation 16 November 1999
Examples

- List books published by Addison-Wesley after 1991, including their year and title.
- Expected result:
  <result>
    <book year="1994">
      <title>TCP/IP Illustrated</title>
    </book>
    <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
    </book>
  </result>
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="2.0">
 <xsl:output method="xml" indent="yes"/>
 <xsl:template match="/">
  <results>
   <xsl:apply-templates select="/bib/book[publisher='Addison-Wesley' and @year>1991]"/>
  </results>
 </xsl:template>
 <xsl:template match="book">
  <book year="{@year}"
   <xsl:copy-of select="title"/>
  </book>
 </xsl:template>
</xsl:stylesheet>
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="2.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="*/">
  <results>
    <xsl:apply-templates select="/bib/book"/>
  </results>
</xsl:template>
<xsl:template match="book[publisher='Addison-Wesley' and @year>1991]">
  <book year="{@year}"/>
  <xsl:copy-of select="title"/>
</book>
</xsl:template>
<xsl:template match="*"/>
</xsl:stylesheet>
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="2.0">
    <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
        <results>
            <xsl:for-each select="/bib/book[publisher='Addison-Wesley' and @year>1991]"/>
            <book year="{@year}">
                <xsl:copy-of select="title"/>
            </book>
        </results>
    </xsl:template>
</xsl:stylesheet>
<results>
  {
    for $b in doc("input.xml")/bib/book
      where $b/publisher = "Addison-Wesley" and $b/@year > 1991
      return
      <book year="{ $b/@year }">
        { $b/title }
      </book>
  }
</results>
fun mkAuthList (val e as (Title, (Author | Editor))*) : (Author, Title)* =
  match e with
  Title[val t], Author[val a], val rest
  -> Author[t], Title[t], mkAuthList(rest)
  | Title[val t], Editor[val e], val rest
  -> mkAuthList(rest)
  | ()
  -> ()
Top-Down Tree Transducer

• An automaton with output

\[ B = \{ Q, \Sigma, \Delta, R, Qs \} \]

• \( Q \) is a finite set of states
• \( \Sigma \) is a finite set of input symbols
• \( \Delta \) is a finite set of output symbols
• \( Qs \) is a start state
• \( R \) is a set of rules of the form
  \[ <\sigma, q> \rightarrow <\delta, q1, q2, ..., qn> \]
Top-Down Tree Transducer Example

- $Q = \{A, B\}$, $\Sigma = \{\langle\text{bib}\rangle, \langle\text{book}\rangle, \langle\text{year}\rangle, \langle\text{author}\rangle, \langle\text{title}\rangle\}$, $\Delta = \{\langle\text{results}\rangle, \langle\text{book}\rangle, \langle\text{year}\rangle, \langle\text{author}\rangle, \langle\text{title}\rangle\}$, $Q_s = \{A\}$
- $R = \{\langle\text{bib}\rangle, A \to \langle\text{result}\rangle (A, B)$
- $\langle\text{book}\rangle, A \to \langle\text{book}\rangle (A, A, A)$
- $\langle\text{book}\rangle, B \to \langle\text{book}\rangle (B, B, B)$
- $\langle\text{year}\rangle, A \to \langle\text{year}\rangle ()$
- $\langle\text{author}\rangle, A \to \langle\text{author}\rangle ()$
- $\langle\text{title}\rangle, A \to \langle\text{title}\rangle ()$
- $\langle\text{year}\rangle, B \to \text{year} ()$
- $\langle\text{author}\rangle, B \to ()$
- $\langle\text{title}\rangle, B \to () \}$
\texttt{<bib>, A -> <result> (A, B)}
<book>, B -> <book> (B, B, B)
<year>, A -> <year> (), <author>, A -> <author> (), <title>, A -> <title> (), <year>, B -> <year> (), <author>, B -> (), <title>, B -> ()
Problems with Top-Down Tree Transducers

- Not powerful enough
- There are several approach to add more power to the transducers
  - Attribute Tree Transducers
  - Macro Tree Transducers
  - k-Pebble Tree Transducers
Macro Tree Transducers

Definition 2.3 A Macro Tree Transducer $M$ is a tuple $(Q, \Sigma, \Delta, e, R)$ with:

(1) a ranked alphabet of states $Q$
(2) a ranked alphabet $\Sigma$ of input symbols
(3) a ranked alphabet $\Delta$ of output symbols, where $\Delta^{(0)} \neq \emptyset$ and $Q \cap (\Sigma \cup \Delta) = \emptyset$
(4) an initial expression $e \in RHS(Q, \Delta, 1, \emptyset)$
(5) a finite set $R$ of rules of the form $q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_r) \rightarrow rhs_{q,\sigma}$
with $k, r \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, $q \in Q^{(r+1)}$ and $rhs_{q,\sigma} \in RHS(Q, \Delta, k, Y_r)$, such that there is exactly one rule for every combination of $q$ and $\sigma$

Example 2.4 Let $Q_{rev} = \{rev^{(2)}\}$ and $\Sigma_{mon} = \{A^{(1)}, B^{(1)}, E^{(0)}\}$. Then we define the Macro Tree Transducer $M_{rev} = (Q_{rev}, \Sigma_{mon}, \Sigma_{mon}, e_{rev}, R_{rev})$ with set of rules $R_{rev}$:

(i) : $rev(A(x), z) \rightarrow rev(x, A(z))$
(ii) : $rev(B(x), z) \rightarrow rev(x, B(z))$
(iii) : $rev(E, z) \rightarrow z$

and $e_{rev} = rev(x_1, E)$.  \(\square\)
Properties of MTT

- $\text{MTT}_1 + \text{MTT}_2 + \text{MTT}_k = k$-pebble $\text{TT}$
- Decidable over MSO if it is of linear size increase
- Express transformation over RTG (no interpretation or comparison of node values)
Type checking problem of transformation

• Approximative approaches (values of nodes are not interpreted!)
Static verification approaches

- Audebaud and Rose: Stylesheet Validation (2000)
  - the initial definition of type-checking problem
  - it propose a solution sketch in form of typing rules on trivial example
- Tozawa: Towards Static Type Checking for XSLT (2001)
  - defines a fragment called XSLT0
  - use an inverse type inference approach
  - FTA -> AFTA
  - completely theoretical
  - RTG + k-pebble transducers
  - shows only one simple XML-QL query
Doug and Bailey: Static Analysis of XSLT Programs (2004)
- Control flow based approach
- Analyse the following properties: reachability, invalid calling relationship, missing templates, termination

Moller at al.: Validation of XSL Transformation (2006)
- control flow based approach (summary graphs)
- focus more on output type validation
Summary of the “State of Art”

As we have seen:

- XML related technologies has a well established mathematical background
- The “type-checking” problem almost completely understand
- No real declarative specification language
- No interpretation of tree values
The Verification problem

- \( P(X) \Rightarrow Q(X, T(X)) \), \( X \) is a tree and \( T \) is a transformation

- How we can describe transformation in high-level manner?

- How we can capture formally the constraints over values of the tree nodes
  - references (IDs)
  - operations over the values (sum, avg, etc.)
Methods

- defining use-cases relevant to practice
- create a high-level specification language
- create a denotational semantic for it
- create a verification calculus
- implementing a verification condition generator
- verification of selected use cases