Static Validation of XSL Transformations
Mads Østerby Olesen
Søren Kuula
Anders Møller
Michael Schwartzbach

Plan
- Brief summary of XSLT (1.0)
- Stylesheet mining
- Type checking XSLT stylesheets
  - simplification
  - flow analysis
  - XML graph construction and validation

XSLT 1.0
- XSLT (XSL Transformations) is designed for stylesheet transformations for document-centric XML languages
- A declarative domain-specific language based on templates and pattern matching using XPath
- An XSLT program consists of template rules, each having a pattern and a template

Processing model
- A source XML tree is transformed by processing its root node
- A single node is processed by
  - finding the template rule with the best matching pattern
  - instantiating its template
    - may create result fragments
    - may select other nodes for processing
- A node list is processed by processing each node and concatenating the results

An example input XML document
```xml
<registrations xmlns="http://eventsRus.org/registrations/">
  <name id="117">John Q. Public</name>
  <group type="private" leader="214">
    <affiliation>Widget, Inc.</affiliation>
    <name id="214">John Doe</name>
    <name id="215">Jane Dow</name>
    <name id="321">Jack Doe</name>
  </group>
  <name id="742">Joe Average</name>
</registrations>
```

An XSLT stylesheet (1/3)
```xml
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:reg="http://eventsRus.org/registrations/"
  xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="reg:registrations">
    <html>
      <head><title>Registrations</title></head>
      <body>
        <ol><xsl:apply-templates/></ol>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="*">
    <li><xsl:value-of select="."/></li>
  </xsl:template>
</xsl:stylesheet>
```
The resulting output

1. John Q. Public
   Widget, Inc.®
   John Doe !!!
2. Jane Dow
3. Jack Doe
4. Joe Average

The challenge

Given
- an XSLT stylesheet $S$
- two schemas, $D_{in}$ and $D_{out}$
assuming that $X$ is valid relative to $D_{in}$
is $S$ applied to $X$ always valid relative to $D_{out}$?

Templates

Main template constructs:
- literal result fragments
  - character data, non-XSLT elements
- recursive processing
  - apply-templates, call-template, for-each, copy, copy-of
- computed result fragments
  - element, attribute, value-of,
- conditional processing
  - if, choose...
- variables and parameters
  - variable, param, with-param

Undecidability

- Since XSLT is Turing complete, this question is clearly undecidable
- Thus, we must apply some approximations
- We want to be able to guarantee correctness
- Thus, we go for conservative approximations (i.e. err on the safe side)
Brief summary of XSLT (1.0)

Stylesheet mining

Type checking XSLT stylesheets
- simplification
- flow analysis
- XML graph construction and validation

How are the many features of XSLT being used?
- typical stylesheet size?
- complexity of select expressions?
- complexity of match expressions?

Obtained via Google:
499 stylesheets with a total of 186,726 lines of code

Stylesheet sizes

Complexity of select expressions

Complexity of match expressions

Plan

- Brief summary of XSLT (1.0)
- Stylesheet mining
- Type checking XSLT stylesheets
  - simplification
  - flow analysis
  - XML graph construction and validation
The XSLT validation algorithm

Our strategy:
1. reduce S to core features of XSLT
2. analyze flow (using \( D_P \))
   - apply-templates \( \rightarrow \) template?
   - possible context nodes when templates are instantiated?
3. construct XML graph
4. validate XML graph relative to \( D_{out} \)

Semantics preserving simplifications
- make defaults explicit (built-in template rules, default select, default axes, coercions, ...)
- insert imported/included stylesheets
- convert literal elements and attributes to element/attribute instructions
- convert text to text instructions
- expand variable uses (not parameters)
- reduce if to choose
- reduce for-each, call-template, and copy to apply-templates instructions and new template rules
- move nested templates (in when/otherwise) to new template rules

Approximating simplifications
- replace each number by a value-of with xslv:unknownString()
- replace each value-of expression by xslv:unknownString(), except for string(self::node()) and string(attribute::a)
- replace when conditions by xslv:unknownBoolean()
- replace name attributes in attribute and element instructions by \( \{xslv:unknownString()\} \), except for constants and \( \{name()\} \)

(We want to handle almost-identity transformations)

Reduced XSLT
The only features left:
- template rules with match, priority, mode, param
- apply-templates with select, mode, sort, with-param
- choose where each condition is xslv:unknownBoolean() and each branch template is an apply-templates
- copy-of with a parameter as argument
- attribute and element whose name is a constant, \( \{name()\} \) or \( \{xslv:unknownString()\} \) and the contents of attribute is a value-of
- value-of where the argument is xslv:unknownString(), string(self::node()) or string(attribute::a)
- top-level param declarations (no variables)

The reduced XSLT stylesheets (1/4)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsv="http://www.brics.dk/XSLTValidator"
    xmlns:xhtml="http://www.w3.org/1999/xhtml"
    version="1.0">

    <template match="*" priority="0">
        <!-- 1 -->
        <element name="out:head">
            <value-of select="string(self::node())"/>
        </element>
        <element name="out:body">
            <value-of select="string(self::node())"/>
        </element>
    </template>
</stylesheet>
```

The reduced XSLT stylesheets (2/4)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsv="http://www.brics.dk/XSLTValidator"
    xmlns:xhtml="http://www.w3.org/1999/xhtml"
    version="1.0">

    <template match="registrations" priority="0">
        <!-- 1 -->
        <template match="in:registrations" priority="0">
            <!-- 1 -->
            <element name="out:html">
                <value-of select="string(self::node())"/>
            </element>
        </template>
    </template>
</stylesheet>
```
The reduced XSLT stylesheet (3/4)

```xml
<template match="in:group" priority="0">
  <element name="out:li">
    <element name="out:table">
      <attribute name="border" select="'1'"/>
      <element name="out:thead">
        <element name="out:tr">
          <element name="out:td">
            <value-of select="in:affiliation"/>
            <choose>
              <when test="xslv:unknownBoolean()">
                <value-of select="'&#174;'"/>
              </when>
              <otherwise/>
            </choose>
          </element>
        </element>
        <apply-templates select="in:name"/>
      </element>
    </element>
  </element>
</template>
```

The reduced XSLT stylesheet (4/4)

```xml
<template match="in:group" priority="0">
  <element name="out:li">
    <element name="out:table">
      <attribute name="border" select="'1'"/>
      <element name="out:thead">
        <element name="out:tr">
          <element name="out:td">
            <value-of select="in:affiliation"/>
            <choose>
              <when test="xslv:unknownBoolean()">
                <value-of select="'&#174;'"/>
              </when>
              <otherwise/>
            </choose>
          </element>
        </element>
        <apply-templates select="in:name"/>
      </element>
    </element>
  </element>
</template>
```

**Plan**

- **Brief summary of XSLT (1.0)**
- Stylesheet mining
- Type checking XSLT stylesheets
  - simplification
  - flow analysis
- XML graph construction and validation

**Flow analysis**

Goals:
- Determine **flow** from `apply-templates` nodes to `template` nodes
- Determine possible **context nodes** for instantiated template nodes

**Flow graphs**

- Define
  - $\Sigma = E \cup (A \times E) \cup \{\text{root, pcdata, comment, pi}\}$
  - $A_S$ = `apply-templates` nodes for $S$
  - $T_S$ = `template` nodes for $S$

- A **flow graph** is a pair $G = (C, F)$ where
  - $C : T_S \rightarrow 2^\Sigma$ describes the **context sets**
  - $F : A_S \times T_S \rightarrow \Sigma \rightarrow 2^\Sigma$ describes the **edge flow**

**A typical situation**

```xml
<template match="/">
  <apply-templates select="child::node()"/>
</template>
```
Fixed point algorithm

Find smallest solution to these constraints:

- \( \text{root} \in C(t) \) if the match expression of \( t \) matches the root
- \( \sigma \in C(t) \Rightarrow \Phi(\sigma, \text{select}_\sigma, \text{match}_t, \text{match}_t) \subseteq F(a, t')(\sigma) \) where \( \Phi(...) \) is an upper approximation of the possible flow from \( a \) in \( t \) to \( t' \) starting with \( \sigma \)
- \( F(a, t)(\sigma) \subseteq C(t) \)

Computing \( \Phi \)

A good version of \( \Phi \) is computed using DFAs:

\[
\sigma' \in \Phi(\sigma, \text{select}_\sigma, \text{match}_t, \text{match}_t) \quad \text{iff} \quad \omega \sigma' \in \Sigma^* R(\alpha) \cap \Sigma^* R(\text{match}_t) \cap \Pi(D_\sigma)
\]

where \( \alpha = \begin{cases} \text{select}_\sigma & \text{if select, starts with } / \\ \text{match}_t / \text{type}(\sigma) / \text{select}_\sigma & \text{otherwise} \end{cases} \)

\( R(x) = \) downwards paths to targets of \( x \)
\( \Pi(D) = \) downwards paths allowed by \( D \)

Example flow graph

Refinements

- Handling modes:
  - delete flow edges with mismatching modes
- Handling priorities:
  - delete flow edges overshadowed by flow edges to higher priority templates
- Handling predicates:
  - use a simple theorem prover (not done yet)

Plan

- Brief summary of XSLT (1.0)
- Stylesheet mining
- Type checking XSLT stylesheets
  - simplification
  - flow analysis
- XML graph construction and validation

XML graphs

- A representation for sets of XML trees
- Different kinds of nodes:
  - element nodes
  - attribute nodes
  - text nodes
  - sequence nodes
  - choice nodes
  
  names and values described by regular string languages
- The language of an XML graph is the set of XML trees obtained by finite unfoldings
### Example XML graph

A graph for `ul` lists with zero or more `li` items:

```
1   E
  2    C
  3    S
  4    S
  5    E
  6    T
      foo
```

- `E`: element
- `A`: attribute
- `T`: text
- `C`: choice
- `S`: sequence

### XML graph fragments

- For each template `t ∈ T_s` and `σ ∈ C(t)` we construct an XML graph fragment describing the possible XML output:
  - the fragment has placeholders for occurrences of `apply-templates` nodes
  - the construction is performed recursively in the template structure (lots of special cases)

### A fragment example

```xml
<element name="out:body">
  <attribute name="bgcolor">
    <value-of select="xslv:unknownString()"/>
  </attribute>
  <element name="out:hr"/>
  Hello!
</element>
```

### Connecting fragments

- The fragments for templates are connected using:
  - the `select` attributes in `apply-templates` nodes
  - the information in the flow graph
  - the information in the input schema

- The challenge is to capture the content model of the output language with sufficient precision

### Default expressions (31.7%)

- It selects all child nodes of the context node `σ`
- Look up the **content model** of `σ` in `D_p`
- Construct an XML graph fragment for this regular language with placeholders for the input elements
- Replace each placeholder `γ` with a choice node leading to the fragments for each template `t'` such that `γ ∈ F(a,t')(σ)`

### Projected contents (44.8%)

- Cases such as:
  - `a`
  - `*`
  - `a | b | c`
- Proceed as for the default case
- But **project** the XML graph fragment onto the chosen sub-alphabet
Multiple location steps (11.9%)

Cases such as
• a/b/c
• /a/b/c

• One step at a time, **concatenate** the fragments

Predicates (3.4%)

Cases such as
• a[...]/b[...]/c[...]
• /a[...]/b[...]/c[...]

• First ignore the predicates
• Then add an edge to an empty sequence to model the case of a false predicate

Attributes (0.6%)

The case @a

• Use $D_a$ to determine whether a is optional

Parent and root (0.4%)

Cases such as:
• ...
• /

• A single node is selected
• Use a choice node with edges to the corresponding outgoing flow

Others (7.2%)

• Use a pessimistic content model containing all sequences of the outgoing flow
• If the possible names are known (5.6%), then use a cardinality analysis (0,1,*+) for more precision

Sorting

• If the *apply-templates* contains *sort* directives, then scramble the determined content model
XML graph validation

- We now have an XML graph that conservatively models the output language.
- We must check that its language is accepted by the output schema $D_{out}$.
- This is done using an algorithm from the JWIG and XACT projects.

Validation errors

A typical error message:

*** Validation error: contents of element 'item' does not match declaration
Rule: <template match="child::*"/>...</template>
Context node: item
Element: <item category="national">...</item>
Schema: (headline,text)

Benchmarks

<table>
<thead>
<tr>
<th>Stylesheet</th>
<th>Lines</th>
<th>Input Schema</th>
<th>Lines</th>
<th>Output Schema</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>xhtml.dtd</td>
<td>104</td>
<td></td>
<td>84</td>
<td></td>
<td>2,278</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>113</td>
<td>xhtml.dtd</td>
<td>56</td>
<td>xhtml.dtd</td>
<td>1,198</td>
</tr>
</tbody>
</table>

Precision

<table>
<thead>
<tr>
<th>Stylesheet</th>
<th>True Errors</th>
<th>False Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>xhtml.dtd</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Efficiency

<table>
<thead>
<tr>
<th>Stylesheet</th>
<th>XG</th>
<th>Flow</th>
<th>Build</th>
<th>Analysis</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>xhtml.dtd</td>
<td>95</td>
<td>0.07</td>
<td>0.07</td>
<td>0.14</td>
<td>0.48</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>22</td>
<td>0.05</td>
<td>0.05</td>
<td>0.10</td>
<td>0.20</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>10</td>
<td>0.06</td>
<td>0.06</td>
<td>0.12</td>
<td>0.24</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>30</td>
<td>0.10</td>
<td>0.06</td>
<td>0.16</td>
<td>0.32</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>12</td>
<td>0.08</td>
<td>0.08</td>
<td>0.16</td>
<td>0.24</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>17</td>
<td>0.12</td>
<td>0.12</td>
<td>0.24</td>
<td>0.36</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>28</td>
<td>0.14</td>
<td>0.14</td>
<td>0.28</td>
<td>0.46</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>34</td>
<td>0.16</td>
<td>0.16</td>
<td>0.32</td>
<td>0.52</td>
</tr>
<tr>
<td>xhtml.dtd</td>
<td>34</td>
<td>0.18</td>
<td>0.18</td>
<td>0.36</td>
<td>0.54</td>
</tr>
</tbody>
</table>

Six unique errors for slideshow.xsl:

*** Validation error: contents of element 'ul' may not match declaration
*** Validation error: required attribute missing in element 'script'
*** Validation error: sub-element 'div' of element 'p' not declared
*** Validation error: sub-element 'text' of element 'div' not declared
*** Validation error: sub-element 'li' of element 'div' not declared
Related work

- Audebaud & Rose 2000:
  - typing rules
  - tiny fragment of XSLT
- Tozawa 2001:
  - inverse type inference (Milo, Suciu, Vianu)
  - even smaller fragment, not implemented
- Dong & Bailey 2004:
  - coarser (but cheaper) flow analysis
  - used for debugging (not static validation)

Recent work

Søren Kuula’s MSc thesis (March 2006):

- full XSLT 2.0 (and thus full XPath 2.0)
- full XML Schema, not just DTD
- much faster:
  - weed out potential flow edges with Dong & Bailey’s technique
  - avoid automata computations as much as possible

Conclusion

- The first, practical validity analyzer for XSLT

Methodology:

- mining to learn about XSLT in practice
- reduce to core features
- pragmatic, conservative approximation
- flow analysis (apply-templates → template)
- XML graphs for validation