Processing, Indexing, and Compression of Structured Data

- XPath queries on XML data streams -

Prof. Dr. Stefan Böttcher
(With slides contributed by Dr. Rita Hartel)
University of Paderborn
EIM – Department of Computer Science
WS 2013/2014
XPath - looking forward (1) - why?

- News ticker (e.g. Reuters News 4 GB/minute)
- Unbounded stream of XML data

- Query: In which newspaper is an article about “Paderborn”
  - Q = //title[. = "Paderborn “]/ancestor::newspaper

- Problem: Information on newspaper is “gone”, when title was read

- Solution idea (by Dan Olteanu et. al.):
  - logical rewrite rules to eliminate reverse-axes from XPath expressions
  - Q’ = /newspaper[//title="Paderborn"]
**XPath - looking forward (2) - idea**

use axes-symmetry

if node type of c is element:

\[
\begin{align*}
  a \in c / \text{ancestor} :: * & \iff c \in a / \text{descendant} :: * \\
  a \in c / \text{ancestor-or-self} :: * & \iff c \in a / \text{descendant-or-self} :: * \\
  p \in c / \text{parent} :: * & \iff c \in p / \text{child} :: *
\end{align*}
\]

\[
\begin{align*}
  p \in c / \text{preceding} :: * & \iff c \in p / \text{following} :: * \\
  p \in c / \text{preceding-sibling} :: * & \iff c \in p / \text{following-sibling} :: *
\end{align*}
\]

More general:

\[
\begin{align*}
  p \in c / \text{reverse-axis} :: * & \iff c \in p / \text{reverse-axis} ^{-1} :: *
\end{align*}
\]

Basic idea:

substitute left-hand side with right-hand-side and adjust the paths
XPath - looking forward (3) - parents

/ descendant :: E1 / parent :: E2

→/ descendant-or-self :: E2
[ ./child :: E1 == /descendant :: E1 ]

( parent⁻¹ = child )

/ descendant :: E1 / ancestor :: E2

→/ descendant-or-self :: E2
[ ./ descendant :: E1 == /descendant :: E1 ]

( ancestor⁻¹ = descendant )
XPath - looking forward (4) - general

/ descendant :: E1 / preceding :: E2

→ / descendant::E2 [./following::E1 == /descendant::E1]

( preceding⁻¹ = following )

/ absolute-path :: E1 / reverse-axis :: E2

→/ descendant-or-self :: E2 [ ./ reverse-axis⁻¹ :: E1 = = /absolute-path :: E1 ]

+ replaces reverse-axis location-step
- introduces a join in the filter
XPath - looking forward (5) - no join

/ descendant :: E1 / preceding :: E2

→ / descendant::E2 [ ./following::E1 == /descendant::E1 ]

( each following :: E1 is a descendant of the root )

+ replaces last reverse-axis location-step
+ avoids the join in the filter
XPath - looking forward (6) - no join

/ descendant :: E1 / ancestor :: E2
→
/descendant-or-self :: E2[./descendant::E1==./descendant::E1]

( each descendant :: E1 is a descendant of the root )

Therefore, this rule can be written without a join:

/ descendant :: E1 / ancestor :: E2
→ / descendant-or-self :: E2 [ . / descendant :: E1 ]

( ancestor ^-1 = descendant )

rule above is applicable to absolute paths only!
XPath - looking forward (7) - no join

/ descendant :: E1 / ancestor :: E2

→ / descendant-or-self :: E2 [ . / descendant :: E1 ]

( ancestor⁻¹ = descendant )

rule above is applicable to absolute paths only!

p / descendant :: E1 / ancestor :: E2

→ p [ descendant :: E1 ] / ancestor :: E2

p / descendant-or-self :: E2 [ . / descendant :: E1 ]
XPath - looking forward (8) - filters

/ descendant :: E1 [ ancestor :: E2 ]
→ / descendant :: E1
[ /descendant-or-self:: E2 / descendant:: node( ) == self::node( ) ]

( ancestor⁻¹ = descendant )

path::E1 [ reverse-axis :: E2 / p2 ]
→ path::E1 [ /descendant-or-self:: E2 [ p2 ] / reverse-axis⁻¹:: node( ) == self::node( ) ]

+ replaces first reverse-axis location-step in the filter
- introduces a join in the filter
To avoid joins, there are 44 special rules to rewrite backward axes, e.g.

- $\text{descendant::n/parent::m} \equiv \text{descendant-or-self::m[child::n]}$
- $\text{child::n/parent::m} \equiv \text{self::m[child::n]}$
- $\text{p/self::n/parent::m} \equiv \text{p[self::n]/parent::m}$
- $\text{p/following-sibling::n/parent::m} \equiv \text{p[following-sibling::n]/parent::m}$
- $\text{p/following::n/parent::m} \equiv \text{p/following::m[child::n]} \mid \text{p/ancestor-or-self::*[following-sibling::n]/parent::m}$
- $\text{descendant::n [parent::m]} \equiv \text{descendant-or-self::m/child::n}$
- $\text{child::n[parent::m]} \equiv \text{self::m/child::n}$
- $\text{p/self::n[parent::m]} \equiv \text{p[parent::m]/self::n}$
- $\text{p/following-sibling::n[parent::m]} \equiv \text{p[parent::m]/following-sibling::n}$
- $\text{p/following::n[parent::m]} \equiv \text{p/following::m/child::n} \mid \text{p/ancestor-or-self::*[parent::m]/following-sibling::n}$
XPath - looking forward (10) - summary

- **Special rules:**
  - either remove a reverse-axis
  - or reduce size of XPath expression with reverse-axis
  - some special rules generate unions ("|") of XPath expressions
  - might lead to an exponential growth in the size of the query

- **General rules:**
  - remove reverse-axes
  - but introduce joins

- XML stream processing possible with the help of query rewriting
Exercise (looking forward)

rewrite the following XPath queries:

- `/a/b/..//c`
- `//a/..//b`

into equivalent queries that use only forward axes:

- transform XPath expressions using the looking forward approach
- Try to avoid using the “special rules”, but try to “derive” the rules (as you know the semantic of the axes)
Evaluating XPath Queries on XML streams: Problem description

Given:
- Infinite XML Data Stream (SAX)
- XPath Query Q

Result:
- Return all XML fragments that match Q in document order
System components

SAX event stream

binary SAX event stream

XPath query

automaton for an XPath query

stack-based query evaluation with reservations for filters
### Reminder: From SAX events to binary SAX events

<table>
<thead>
<tr>
<th>Pairs of SAX-Events</th>
<th>Location Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>end-element( _)</td>
<td>next-sibling :: a</td>
</tr>
<tr>
<td>start-element( a )</td>
<td></td>
</tr>
<tr>
<td>start-element( a )</td>
<td>first-child :: a</td>
</tr>
<tr>
<td>end-element( _ )</td>
<td>parent :: *</td>
</tr>
<tr>
<td>end-element( _ )</td>
<td></td>
</tr>
<tr>
<td>start-element( _ )</td>
<td>no location step</td>
</tr>
<tr>
<td>end-element( _ )</td>
<td></td>
</tr>
</tbody>
</table>
XPath query normalization

1. Remove all backward axes,
e.g. Q = / a / b / parent::* / c → Q´ = /a[b]/c

2. Remove following-axes
following::* →
ancestor-or-self::* / following-sibling::* / descendant-or-self::*

3. Remove all backward axes that were introduced in the previous step

Result:
equivalent XPath query with only few forward-axes
Automata for XPath location steps

/child::a

/child::a

/descendant::a

/descendant-or-self::a

/following-sibling::a

/self::a
Gluing atomic XPath automata together to an XPath automaton

Query \( Q = /\text{child}::a/\text{child}::c \)

automaton for \( /a/c \) is combined from automata for \( /a \) and for \( ./c \)
Evaluating filter-free path queries

Event

Stack: {0}
Evaluating filter-free path queries

Event | Stack
--- | ---
first-child::a | {0} | {1,2}
Evaluating filter-free path queries

Event | Stack: {0}
-------|--------
first-child::a | {0} | {1,2}
first-child::b | {0} | {1,2} | {3}
Evaluating filter-free path queries

Event  Stack: {0}

first-child::a  {0} | {1,2}
first-child::b  {0} | {1,2} | {3}
parent::*  {0} | {1,2}
Evaluating filter-free path queries

Event | Stack: \{0\}
---|---
first-child::a | \{0\} | \{1,2\}
first-child::b | \{0\} | \{1,2\} | \{3\}
parent::* | \{0\} | \{1,2\} | \xmark\{3\}
next-sibling::a | \{0\} | \{1,2\}
Evaluating filter-free path queries

<table>
<thead>
<tr>
<th>Event</th>
<th>Stack:</th>
<th>{0}</th>
<th>{1,2}</th>
<th>{3}</th>
</tr>
</thead>
<tbody>
<tr>
<td>first-child::a</td>
<td></td>
<td>{0}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>first-child::b</td>
<td></td>
<td>{0}</td>
<td>{1,2}</td>
<td></td>
</tr>
<tr>
<td>parent::*</td>
<td></td>
<td>{0}</td>
<td>{1,2}</td>
<td></td>
</tr>
<tr>
<td>next-sibling::a</td>
<td></td>
<td>{0}</td>
<td>{1,2}</td>
<td></td>
</tr>
<tr>
<td>first-child::c</td>
<td></td>
<td>{0}</td>
<td>{1,2}</td>
<td>{3}</td>
</tr>
</tbody>
</table>
Evaluating filter-free path queries

Event | Stack: {0} | Result
--- | --- | ---
first-child::a | {0} | {1,2} | stop output of result
first-child::b | {0} | {1,2} | {3}
parent::* | {0} | {1,2} | {3}
next-sibling::a | {0} | {1,2} | {3}
first-child::c | {0} | {1,2} | {3}
next-sibling::b | {0} | {1,2} | {3}
Evaluating filter-free path queries

0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4

Event | Stack: \{0\}
--- | ---
first-child::a | \{0\} | \{1,2\}
first-child::b | \{0\} | \{1,2\} | \{3\}
pARENT::* | \{0\} | \{1,2\}
next-sibling::a | \{0\} | \{1,2\}
first-child::c | \{0\} | \{1,2\} | \{3,4\}
next-sibling::b | \{0\} | \{1,2\} | \{3\}
next-sibling::c | \{0\} | \{1,2\} | \{3,4\}

start output
of result
## Evaluating filter-free path queries

<table>
<thead>
<tr>
<th>Event</th>
<th>Stack: {0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>first-child::a</td>
<td>{0}</td>
</tr>
<tr>
<td>first-child::b</td>
<td>{0}</td>
</tr>
<tr>
<td>parent::*</td>
<td>{0}</td>
</tr>
<tr>
<td>next-sibling::a</td>
<td>{0}</td>
</tr>
<tr>
<td>first-child::c</td>
<td>{0}</td>
</tr>
<tr>
<td>next-sibling::b</td>
<td>{0}</td>
</tr>
<tr>
<td>next-sibling::c</td>
<td>{0}</td>
</tr>
<tr>
<td>parent::*</td>
<td>{0}</td>
</tr>
</tbody>
</table>

Event stop

Output of result
Evaluating filter-free path queries

Event | Stack: {0}
-------|------------------
first-child::a | {0} | {1,2}
first-child::b | {0} | {1,2} | {3}
parent::* | {0} | {1,2}
next-sibling::a | {0} | {1,2}
first-child::c | {0} | {1,2} | {3,4}
next-sibling::b | {0} | {1,2} | {3}
next-sibling::c | {0} | {1,2} | {3,4}
prefix:::* | {0} | {1,2} | \(\times\) {3,4}
prefix:::* | {0} | {1,2}
prefix:::* | {0}
Evaluating path queries with predicate filters

\[ Q = /a[b]/c \]

Event | Stack: {0}
Evaluating path queries with predicate filters

Q = /a[b]/c

Event        Stack: {0}
first-child::a {0} | {1,2-R1, R1:{5}}
Evaluating path queries with predicate filters

Q = /a[b]/c

Event | Stack: {0}
--- | ---
first-child::a | \{0\} | \{1,2-R1,R1:{5}\}
first-child::b | \{0\} | \{1,2-R1,R1:{5}\} | \{3-R1,R1:{6,7}\}
Evaluating path queries with predicate filters

Q = /a[b]/c

Event | Stack: {0}
--- | ---
first-child::a | {0} | {1,2} ,R1:{5}}
first-child::b | {0} | {1,2} ,R1:{5}} | {3 ,R1:{6,7}}
Evaluating path queries with predicate filters

Q = /a[b]/c

Event | Stack
--- | ---
first-child::a | {0} | {1,2} ,R1:{5}}
first-child::b | {0} | {1,2} ,R1:{5}} | {3} ,R1:{6,7}}
pARENT:: | {0} | {1,2} ,R1:{5}}
Evaluating path queries with predicate filters

Q = /a[b]/c

<table>
<thead>
<tr>
<th>Event</th>
<th>Stack:</th>
<th>0(\rightarrow)1(\rightarrow)2(\rightarrow)3(\rightarrow)4(\rightarrow)5(\rightarrow)6(\rightarrow)7</th>
</tr>
</thead>
<tbody>
<tr>
<td>first-child::a</td>
<td>{0}(\rightarrow){1,2},R1:{5}}</td>
<td></td>
</tr>
<tr>
<td>first-child::b</td>
<td>{0}(\rightarrow){1,2},R1:{5}} (\rightarrow){3},R1:{6,7}}</td>
<td></td>
</tr>
<tr>
<td>parent::*</td>
<td>{0}(\rightarrow){1,2},R1:{5}}</td>
<td></td>
</tr>
<tr>
<td>next-sibling::a</td>
<td>{0}(\rightarrow){1,2-R2,R2:{5}}</td>
<td></td>
</tr>
</tbody>
</table>
Evaluating path queries with predicate filters

Q = /a[b]/c

Event | Stack: {0}
--- | ---
first-child::a | {0} | {1,2,R1:[5]}
first-child::b | {0} | {1,2,R1:[5]} | {3,R1:[6,7]}
parent::* | {0} | {1,2,R1:[5]}
next-sibling::a | {0} | {1,2-R2,R2:[5]}
first-child::c | {0} | {1,2-R2,R2:[5]} | {3-R2,4-R2,R2:[6]}

start store results of c
Evaluating path queries with predicate filters

Q = /a[b]/c

Event | Stack: {0}
--- | ---
first-child::a | {0} | {1,2,R1:{5}}
first-child::b | {0} | {1,2,R1:{5}} | {3,R1:{6,7}}
parent::* | {0} | {1,2,R1:{5}}
next-sibling::a | {0} | {1,2-R2,R2:{5}}
first-child::c | {0} | {1,2-R2,R2:{5}} | {3-R2,R2:{6}}
next-sibling::b | {0} | {1,2-R2,R2:{5}} | {3-R2,R2:{6,7}}

stop store results of c
Evaluating path queries with predicate filters

Q = /a[b]/c

<table>
<thead>
<tr>
<th>Event</th>
<th>Stack: {0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>first-child::a</td>
<td>{0}</td>
</tr>
<tr>
<td>first-child::b</td>
<td>{0}</td>
</tr>
<tr>
<td>parent::*</td>
<td>{0}</td>
</tr>
<tr>
<td>next-sibling::a</td>
<td>{0}</td>
</tr>
<tr>
<td>first-child::c</td>
<td>{0}</td>
</tr>
<tr>
<td>next-sibling::b</td>
<td>{0}</td>
</tr>
</tbody>
</table>

output of c result
filter is satisfied
Evaluating path queries with predicate filters

\[ Q = /a[b]/c \]

<table>
<thead>
<tr>
<th>Event</th>
<th>Stack: {0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>first-child::a</td>
<td>{0}</td>
</tr>
<tr>
<td>first-child::b</td>
<td>{0}</td>
</tr>
<tr>
<td>parent::*</td>
<td>{0}</td>
</tr>
<tr>
<td>next-sibling::a</td>
<td>{0}</td>
</tr>
<tr>
<td>first-child::c</td>
<td>{0}</td>
</tr>
<tr>
<td>next-sibling::b</td>
<td>{0}</td>
</tr>
<tr>
<td>next-sibling::c</td>
<td>{0}</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation time for Q5 on different document sizes

Evaluation time for different queries on smallest and on largest document size

Evaluation time for Q5 on different document sizes
Summary

XPath query

/a/c

SAX event stream

automaton for an XPath query

stack-based query evaluation with reservations for filters

Processing, Indexing, and Compression of Structured Data – WS 2013/2014 – XPath queries on XML streams
Processing XPath queries on XML streams (summary)

Abonnement for XPath queries using forward-axes only

- implemented by automata
  - states describe on which XPath path starting at the root the current context node occurs
  - edges represent pairs SAX parser events (i.e. pairs of opening and closing element tags detected)
  - edge labels (next-sibling or first child) describe elementary location steps performed by pair of SAX parser events that change the state of the automaton

- Not applicable to reverse-axes (parent, ancestor, preceding(-sibling))
Conclusions