Valid Query Answers for XML

Slawomir Staworko\textsuperscript{1}  Jan Chomicki\textsuperscript{2}

\textsuperscript{1}University at Buffalo and INRIA Futurs
\textsuperscript{2}University at Buffalo and Warsaw University

June 25-29, 2007
Invalid XML documents

Querying Invalid XML

- Integration of XML documents
- Slight differences between schemas (e.g. different cardinality constraints)
- Legacy XML databases
- Database updates
Querying Invalid XML

- Integration of XML documents
- Slight differences between schemas (e.g. different cardinality constraints)
- Legacy XML databases
- Database updates

Document Type Definition ($D_0$)

proj $\rightarrow$ (name, emp, proj*, emp*)
emp $\rightarrow$ (name, salary)
name $\rightarrow$ #PCDATA
salary $\rightarrow$ #PCDATA

Query: get salaries of all employees that are not managers

//proj/name/emp/following-sibling::emp/salary
**Invalid XML documents**

- Integration of XML documents
- Slight differences between schemas (e.g. different cardinality constraints)
- Legacy XML databases
- Database updates

**Document Type Definition ($D_0$)**

- $proj \rightarrow (name, emp, proj^*, emp^*)$
- $emp \rightarrow (name, salary)$
- $name \rightarrow \#PCDATA$
- $salary \rightarrow \#PCDATA$

**Query:** get salaries of all employees that are not managers

```
//proj/name/emp/following-sibling::emp/salary
```
Invalid XML documents

Querying Invalid XML
- Integration of XML documents
- Slight differences between schemas (e.g. different cardinality constraints)
- Legacy XML databases
- Database updates

Document Type Definition (D₀)

proj → (name, emp, proj*, emp*)
emp → (name, salary)
name → #PCDATA
salary → #PCDATA

Document with errors

proj
  name Pierogies
  emp
    name John
    salary $80k
  emp
    name Mary
    salary $40k
  ...

Query: get salaries of all employees that are not managers

//proj/name/emp/following-sibling::emp/salary
Invalid XML documents

Querying Invalid XML

- Integration of XML documents
- Slight differences between schemas (e.g. different cardinality constraints)
- Legacy XML databases
- Database updates

Document Type Definition ($D_0$)

```xml
proj → (name, emp, proj*, emp*)
emp → (name, salary)
name → #PCDATA
salary → #PCDATA
```

Query: get salaries of all employees that are not managers

```
//proj/name/emp/following-sibling::emp/salary
```
Core XPath Queries

- text values but **no attributes**
- all standard axes
- subexpressions, value tests, and joins: //*[A/B], //*[B/text()=C/text()]
- negation and disjunction: //*[not A/B], //A | //B
- no functions

count(/A/*)

Tree Reachability Fact: \((x, Q, y)\)

Basic facts use only \(\ast\) and **following-sibling::**:

\((N_0,\ast, N_1), (N_1,\ast, N_2),...\)

Other facts are **inferred** with implications:

\((X, Q/P, Y) \iff (X, Q, Z) \land (Z, P, Y).\)

\((N_0,\ast/\ast, N_2).\)

Query Answers

- given query \(Q\) and document \(T\) with the root node \(r\)
- find all tree facts that hold in \(T\)
- \(x\) in an answer to \(Q\) in \(T\) iff the tree fact \((r, Q, x)\) holds in \(T\)
Query Evaluation for Positive Core XPath (no negation)

Bottom-up approach

- computing tree facts for query $Q$
- tree facts for $T_1, \ldots, T_m$ computed before
- including inferred facts (involving subqueries of $Q$)

Tree

```
X
  ↓
 X_{i-1}
  ↓
 X_i
  ↓
 X_m
```

Algorithm

1. start with $\emptyset$
2. for subtree $T_i$ ($i = 1, \ldots, m$)
   1. add all facts of the subtree (obtained by recursion)
   2. add $(N_0,/*,N_i)$
   3. if $i > 1$ add $(N_{i-1},following_sibling::*,N_i)$
Editing operations

- Insert: Cost: 3
- Delete: Cost: 2
- Modify: Cost: 1
Editing operations

- Inserting a subtree
  - Cost: 3
  - Insert
  - Cost: 2
  - Modify
  - Cost: 1
Editing operations

- Inserting a subtree
  - Cost: 3
- Deleting a subtree
  - Cost: 2
Editing operations

- Inserting a subtree
- Deleting a subtree
- Modifying a node's label
**Distance between documents**

\[ \text{dist}(T, S) \] is the minimal cost of transforming \( T \) into \( S \)

**Distance to a DTD**

\[ \text{dist}(T, D) \] is the minimal cost of repairing \( T \) w.r.t \( D \) i.e.,

\[ \min \{ \text{dist}(T, S) \mid S \text{ valid w.r.t } D \} \]

**DTD**

- \( C \rightarrow (A, B) \)
- \( A \rightarrow \text{EMPTY} \)
- \( B \rightarrow \text{EMPTY} \)

**Repair**

\( T' \) is a repair of \( T \) w.r.t \( D \) iff

\[ \text{dist}(T', T) = \text{dist}(T, D) \]

There can be an exponential number of repairs.
Valid Query Answers

\[ x \] is a valid answer to query \( Q \) in \( T \) w.r.t. \( D \) iff
\[ x \] is an answer to \( Q \) in every repair of \( T \) w.r.t. \( D \).

XML Document

```
proj -> (name, emp, proj*, emp*)
emp -> (name, salary)
name -> #PCDATA
salary -> #PCDATA
```

Queries and Valid Answers

\[
\begin{align*}
//\text{proj[emp[1]/salary='} & \text{"90K"}/name/text() \rightarrow \{\text{Pierogies}\} \\
//\text{proj[name='Pierogies']/emp[1]/salary/text()} \rightarrow \{\text{"90K"}\} \\
//\text{proj[name='Pierogies']/emp[1]/name/text()} \rightarrow \emptyset \\
//\text{proj[name='Pierogies']/emp[1]/salary} \rightarrow \emptyset
\end{align*}
\]
Trace graph

DTD

C → (A,B)*
A → EMPTY
B → EMPTY

Q₀ → Q₁

Read A → Read B

A B B

A B B

A B
Trace graph

DTD

C → (A,B)*
A → EMPTY
B → EMPTY

A
B

Q₀
Q₁

Del
Ins A
Ins B
Read B
Read A
Del
Trace graph

**DTD**

\[
\begin{align*}
C & \rightarrow (A,B)^* \\
A & \rightarrow \text{EMPTY} \\
B & \rightarrow \text{EMPTY}
\end{align*}
\]

**Trace Graph**

- **Nodes**: \(Q_0\) and \(Q_1\)
- **Transitions**:
  - \(Q_0\) to \(Q_0\) on \(\text{Read A}\)
  - \(Q_0\) to \(Q_1\) on \(\text{Ins A}\)
  - \(Q_1\) to \(Q_0\) on \(\text{Del}\)
  - \(Q_1\) to \(Q_1\) on \(\text{Ins B}\)
  - \(Q_0\) to \(Q_1\) on \(\text{Read B}\)
  - \(Q_1\) to \(Q_0\) on \(\text{Del}\)

**Trace Paths**:

- \(\text{Read, Read, Del}\)
- \(\text{Read, Read, Ins A, Read}\)
- \(\text{Read, Del, Read}\)
Trace graph

**DTD**

- C → (A, B)*
- A → EMPTY
- B → EMPTY

**Trace graph**
Trace graph

DTD

C → (A,B)*
A → EMPTY
B → EMPTY

A

B

C

Q₀

Q₁

Q₀ → Del
Q₀ → Del
Q₀ → Del
Q₀ → Del

Q₁ → Del
Q₁ → Del
Q₁ → Del
Q₁ → Del
Trace graph

**Trace Graph**

- **Nodes**: Q0, Q1
- **Edges**:
  - Q0,0 → Q0,1
  - Q0,1 → Q0,0
  - Q0,0 → Q0,1
  - Q0,1 → Q0,0
  - Q1,0 → Q1,1
  - Q1,1 → Q1,0
  - Q1,1 → Q1,2
  - Q1,2 → Q1,1

**Actions**:
- Ins A
- Ins B
- Read A
- Read B
- Del

**DTD**
- C → (A, B)*
- A → EMPTY
- B → EMPTY

**Diagram**
- States: Q0, Q1
- Transitions: Ins A, Ins B, Read A, Read B, Del

**Example**: Repairs:
- (Read, Read, Del)
- (Read, Read, Ins A, Read)
- (Read, Del, Read)
Trace graph

DTD

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>(A,B)*</td>
</tr>
<tr>
<td>A</td>
<td>\text{EMPTY}</td>
</tr>
<tr>
<td>B</td>
<td>\text{EMPTY}</td>
</tr>
</tbody>
</table>

A → \text{EMPTY}  
B → \text{EMPTY}  

Trace graph:

- Q₀
- Q₁

Q₀ → Q₁, Q₁ → Q₀

Q₀ → Q₀, Del  
Q₀ → Q₁, Ins A

Q₁ → Q₀, Del  
Q₁ → Q₁, Ins A

Q₀ → Q₀, Del  
Q₀ → Q₁, Ins A

Q₁ → Q₀, Del  
Q₁ → Q₁, Ins A

Q₀ → Q₀, Del  
Q₀ → Q₁, Ins A

Q₁ → Q₀, Del  
Q₁ → Q₁, Ins A
Trace graph

Compact representation of all repairs

DTD

\[
\begin{align*}
C & \rightarrow (A,B)^* \\
A & \rightarrow \text{EMPTY} \\
B & \rightarrow \text{EMPTY}
\end{align*}
\]

Repairing Paths:

- \((\text{Read, Read, Del})\)
- \((\text{Read, Read, Ins A, Read})\)
- \((\text{Read, Del, Read})\)
Computing Valid Query Answers

**Certain Tree Facts**
- Tree facts present in every repair of a given tree

**Bottom-up approach**
- Precomputed values:
  - certain facts for all children
  - certain facts common for every minimal tree satisfying DTD

**Obtain certain facts by**
- Intersection of the sets of all repairing paths

For every repairing path, construct:
- set of facts collected “so far”:
  - start with $\emptyset$
  - **Read** adds certain facts of the corresponding child
  - **Del** adds nothing
  - **Ins A** adds certain facts common for minimal trees labeled with A
Eager intersection

Problem
Possibly an exponential number of paths

Solution: Eager Intersection
Intersect all sets of certain facts for paths sharing the same last operation adding facts (Read/Ins).

Computing VQA

<table>
<thead>
<tr>
<th>Queries</th>
<th>Combined-complexity</th>
<th>Data-complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descending axes</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
<tr>
<td>+ Ascending axes</td>
<td>co-NP-complete</td>
<td>???</td>
</tr>
<tr>
<td>+ Sliding axes</td>
<td>co-NP-complete</td>
<td>???</td>
</tr>
<tr>
<td>+ Negation/Disjunction</td>
<td>co-NP-complete</td>
<td>???</td>
</tr>
<tr>
<td>+ Joins</td>
<td>co-NP-complete</td>
<td>co-NP-complete</td>
</tr>
</tbody>
</table>
Experimental Results: Edit Distance Computation

Compared algorithms

- **PARSE** base line
- **VALIDATE** regular automata
- **DIST** distance computation

Data generation

1. random valid document
2. removing and adding random nodes
3. invalidity ratio
   \[ \text{dist}(T, D)/|T| \simeq 0.1\% \]
4. small height (8-10)
Experimental Results: Edit Distance Computation

**Compared algorithms**

- **PARSE**: base line
- **VALIDATE**: regular automata
- **DIST**: distance computation

**Data generation**

1. random valid document
2. removing and adding random nodes
3. invalidity ratio
   \[ \text{dist}(T, D)/|T| \approx 0.1\% \]
4. small height (8-10)
Experimental Results: Edit Distance Computation

**Compared algorithms**

- **PARSE**: base line
- **VALIDATE**: regular automata
- **DIST**: distance computation

**Data generation**

1. random valid document
2. removing and adding random nodes
3. invalidity ratio
   \[ \text{dist}(T, D)/|T| \approx 0.1\% \]
4. small height (8-10)
Experimental Results: Valid Query Answer Computation

**Compared algorithms**

<table>
<thead>
<tr>
<th>QA</th>
<th>base line</th>
</tr>
</thead>
<tbody>
<tr>
<td>VQA</td>
<td>eager intersection</td>
</tr>
</tbody>
</table>

**Data generation**

- invalidity ratio
  
  \[ \frac{\text{dist}(T, D)}{|T|} \approx 0.1\% \]

- small height (8-10)

**Implemented Queries**

- `//, /, following_sibling::`
- `name()=A,text()='str'`
- QA works in linear time
Experimental Results: Valid Query Answer Computation

**Compared algorithms**

- QA  base line
- VQA  eager intersection

**Data generation**

- invalidity ratio
  \[ \text{dist}(T,D)/|T| \approx 0.1\% \]
- small height (8-10)

**Implemented Queries**

- `//, /, following_sibling::`
- `name()='A',text()='str'`
- QA works in linear time

---

**Graph**

- QA
- VQA

**Axes**

- Document size (MB)
- Time (sec.)
Experimental Results: Valid Query Answer Computation

Compared algorithms
- QA base line
- VQA eager intersection

Data generation
- invalidity ratio
  \[ \text{dist}(T, D)/|T| \approx 0.1\% \]
- small height (8-10)

Implemented Queries
- `//, /, following_sibling::`
- `name()=A,text()='str'`
- QA works in linear time

![Graph showing the relationship between DTD size and query answer computation time. The line labeled VQA indicates the performance of the VQA algorithm.]
Conclusions and Future Work

Conclusions

- Framework for querying documents with validity violations of local nature (missing or superfluous nodes)
- Efficient algorithm for computing valid answers to a class of XPath queries

Future Work

- Valid answers by query rewriting
- In-depth analysis of data complexity
- Other tree operations: inner node deletion/insertion, node move, . . . [1]
- Semantic inconsistencies: keys, functional dependencies,. . . [2]
Acknowledgments

Marcelo Arenas
Leo Bertossi
Wenfei Fan
Jerzy Marcinkowski
Slawomir Staworko
Jef Wijsen
