Validity-Sensitive Querying of XML Databases

Slawomir Staworko   Jan Chomicki

Department of Computer Science and Engineering
University at Buffalo, SUNY

Motivation

Querying Invalid XML

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

Motivation

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)

$$
\begin{aligned}
\text{proj} &\longrightarrow (\text{name, emp, proj*, emp*}) \\
\text{emp} &\longrightarrow (\text{name, salary}) \\
\text{name} &\longrightarrow \#\text{PCDATA} \\
\text{salary} &\longrightarrow \#\text{PCDATA}
\end{aligned}
$$

Query: get salaries of all employees that are not managers

//proj/name/emp/following_sibling::emp/salary

Motivation

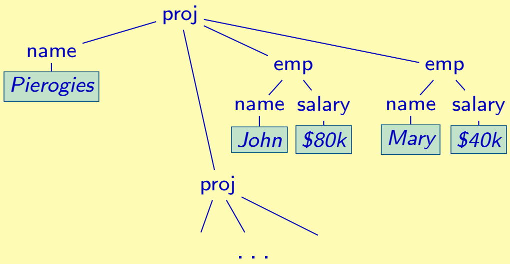
Querying Invalid XML

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

Document Type Definition (D_0)

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

Document with errors



Query: get salaries of all employees that are not managers

//proj/name/emp/following_sibling::emp/salary

Motivation

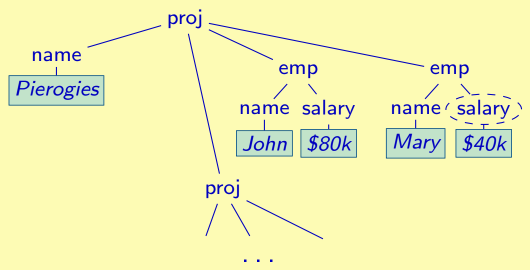
Querying Invalid XML

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

Document Type Definition (D_0)

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

Document with errors



Query: get salaries of all employees that are not managers

//proj/name/emp/following_sibling::emp/salary

Motivation

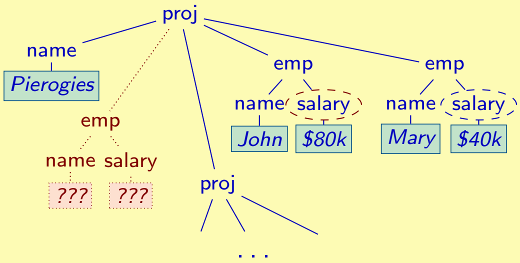
Querying Invalid XML

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

Document Type Definition (D_0)

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

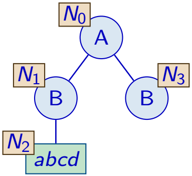
Document with errors



Query: get salaries of all employees that are not managers

//proj/name/emp/following_sibling::emp/salary

Positive Core XPath



Positive Core XPath Queries

- text values but without attributes
- all standard axes
- tests with subexpressions and equality:
  `//*[A/B]`, `//*[B//text()=‘abcd’]`
- but no negation in tests:
  `//*[not A/B]`, `//*[B/text()≠‘abcd’]`
- and no functions

Tree Reachability Fact: (x, Q, y)

$$\text{node } x \xrightarrow{\text{query } Q} \text{object } y$$

Basic facts use only `/*` and `following_sibling::`

$$(N_0, /*, N_1),\ (N_1, /*, N_2)$$

Other facts are inferred with (Horn) rules
$(X, Q/P, Y) \Leftarrow (X, Q, Z) \wedge (Z, P, Y)$.

$$(N_0, /*/*, N_2).$$

Query Answers

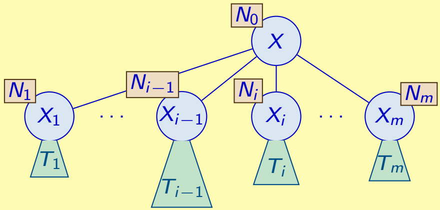
- given query Q and document T with the root node r
- find all tree facts that hold in T
- x in an answer to Q in T iff the tree fact (r, Q, x) holds in T

Query Evaluation

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



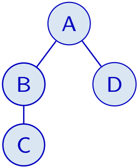
Algorithm

I start with $\varnothing$

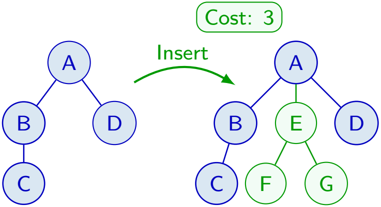
II for subtree T_i ($i = 1, \ldots, m$)

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



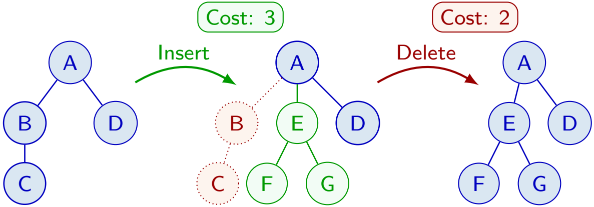
Editing operations



Editing operations

- Inserting a subtree

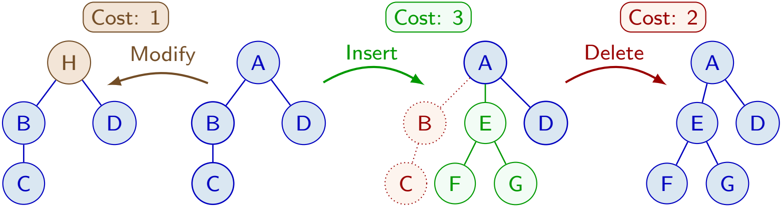
Editing operations



Editing operations

- Inserting a subtree
- Deleting a subtree

Editing operations



Editing operations

- Inserting a subtree
- Deleting a subtree
- Modifying node's label

Edit Distance and Repairs

Distance between documents

$dist(T, S)$ is the minimal cost of transforming T into S

Distance to a DTD

$dist(T, D)$ is the minimal cost of repairing T w.r.t D i.e.,

$$\min\{dist(T, S) | S \text{ valid w.r.t } D\}$$

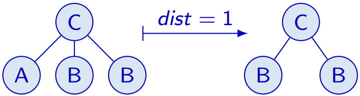
DTD

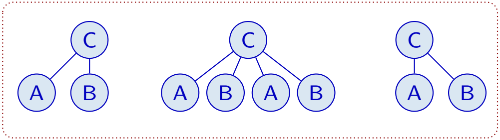
C $\rightarrow$ (A,B)*
A $\rightarrow$ EMPTY
B $\rightarrow$ EMPTY

Repair

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

$$dist(T', T) = dist(T, D)$$





There can be an exponential number of repairs

Valid Query Answers

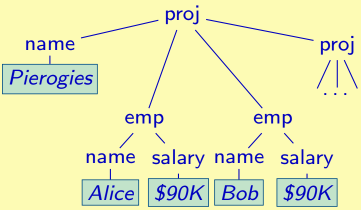
Valid Query Answers

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.

Document Type Definition (D_0)

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

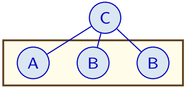
XML Document



Queries and Valid Answers

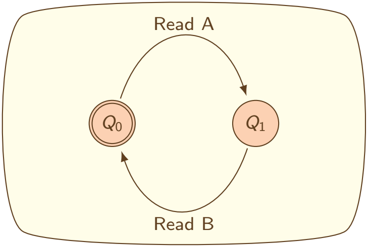
`//proj[emp[1]/salary='$90K']/name/text()`	$\longrightarrow$ {*Pierogies*}
`//proj[name='Pierogies']/emp[1]/salary/text()`	$\longrightarrow$ {*$90K*}
`//proj[name='Pierogies']/emp[1]/name/text()`	$\longrightarrow \emptyset$
`//proj[name='Pierogies']/emp[1]/salary`	$\longrightarrow \emptyset$

Trace graph

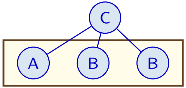


DTD

C → (A,B)*
A → EMPTY
B → EMPTY

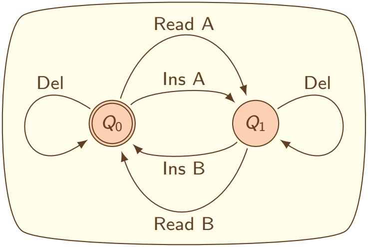


Trace graph

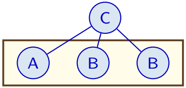


DTD

C → (A,B)*
A → EMPTY
B → EMPTY

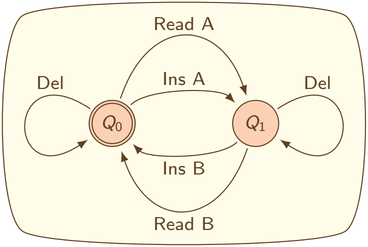


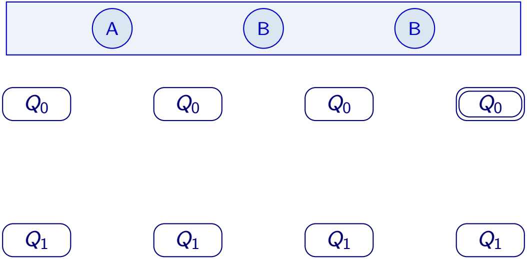
Trace graph



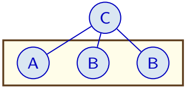
DTD

C → (A,B)*
A → EMPTY
B → EMPTY



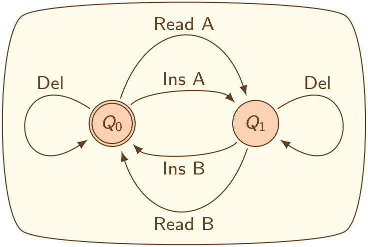


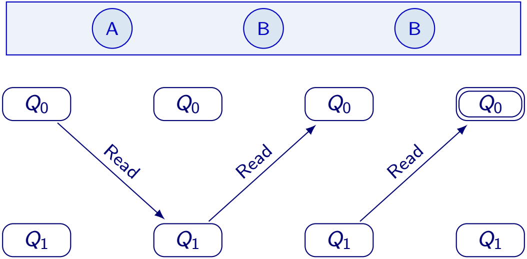
Trace graph



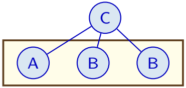
DTD

C → (A,B)*
A → EMPTY
B → EMPTY



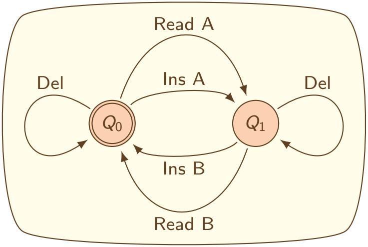


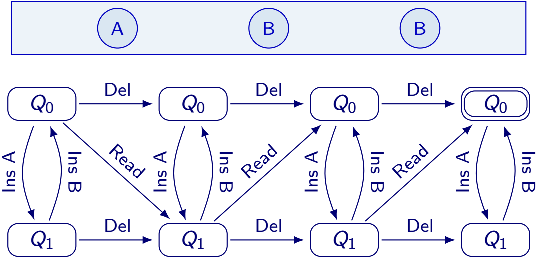
Trace graph



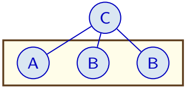
DTD

C → (A,B)*
A → EMPTY
B → EMPTY



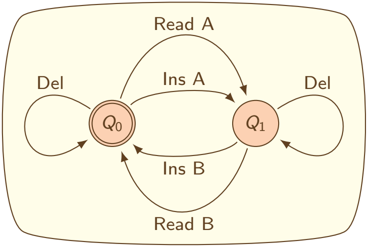


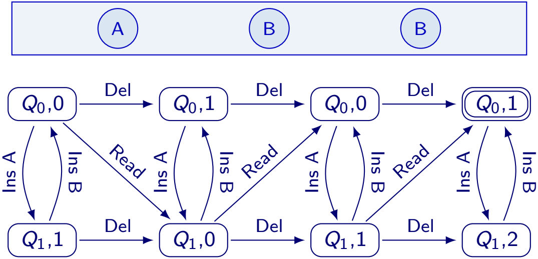
Trace graph



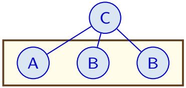
DTD

C → (A,B)*
A → EMPTY
B → EMPTY



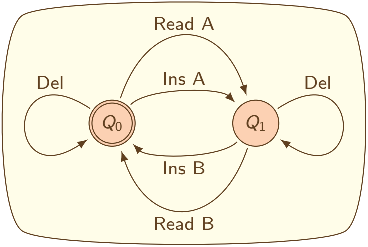


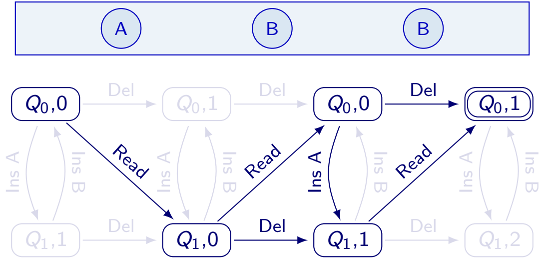
Trace graph



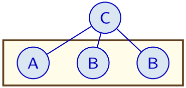
DTD

C → (A,B)*
A → EMPTY
B → EMPTY



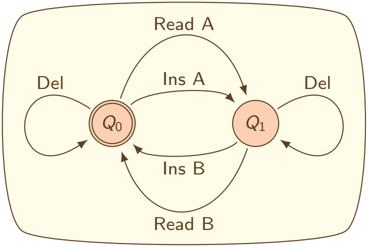


Trace graph

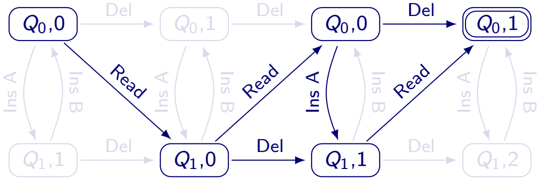


DTD

C → (A,B)*
A → EMPTY
B → EMPTY



Compact representation of all repairs



Repairing Paths:

- (Read, Read, Del)
- (Read, Read, Ins A, Read)
- (Read, Del, Read)

Computing Valid Query Answers

Certain Tree Facts

Tree facts present in every repair of a given tree

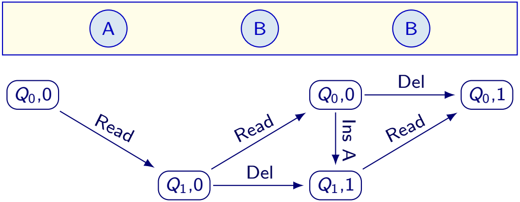
Bottom-up approach

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 $\varnothing$
- **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

For queries without tests of form (joins)

```
[Q/text() = P/text()].
```

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

Data complexity of VQA

Computation of valid answers to positive core XPath queries without joins can be performed in polynomial time in the size of the document.

Data complexity of VQA with joins

There exists a query with joins for which computing VQA is co-NP-complete.

Combined complexity of VQA

Combined complexity of computing valid query answers is co-NP-complete.

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

   $$dist(T, D)/|T| \simeq 0.1\%$$

4. small height (8-10)

Experimental Results: Edit Distance Computation

Compared algorithms

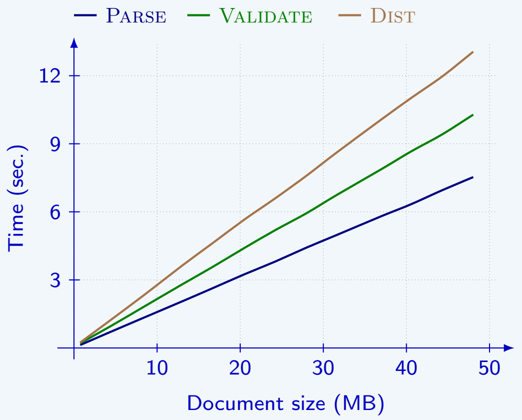
PARSE	base line
VALIDATE	regular automata
DIST	distance computation

Data generation

1. random valid document
2. removing and adding random nodes
3. invalidity ratio

   $$dist(T, D)/|T| \simeq 0.1\%$$

4. small height (8-10)



Experimental Results: Edit Distance Computation

Compared algorithms

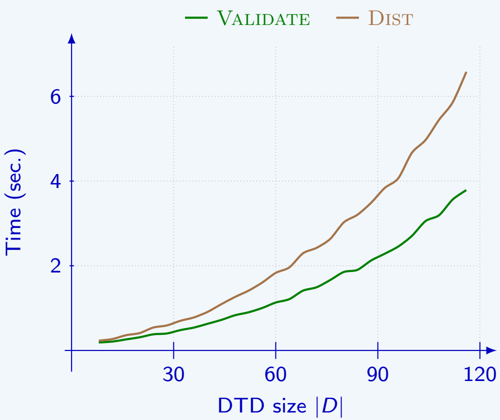
Parse	base line
Validate	regular automata
Dist	distance computation

Data generation

1. random valid document
2. removing and adding random nodes
3. invalidity ratio

   $$dist(T, D)/|T| \simeq 0.1\%$$

4. small height (8-10)



Experimental Results: Valid Query Answer Computation

Compared algorithms

QA	base line
VQA	eager intersection

Data generation

- invalidity ratio

$$dist(T, D)/|T| \simeq 0.1\%$$

- 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

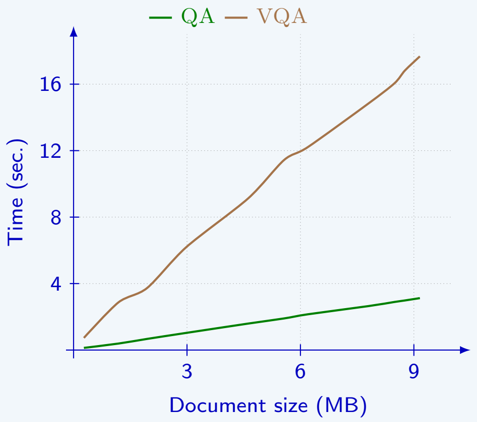
- invalidity ratio

  $$dist(T, D)/|T| \simeq 0.1\%$$

- 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

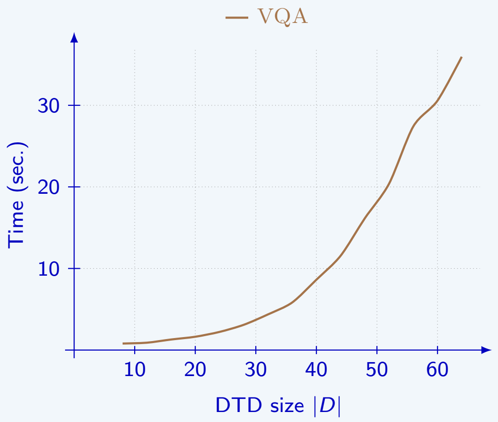
- invalidity ratio

  $$dist(T, D)/|T| \simeq 0.1\%$$

- small height (8-10)

Implemented Queries

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



Conclusions and Future Work

Conclusions

- Framework for querying of 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
- Valid answers to queries with negation
- Other tree operations (swap, shift, contraction, . . .)
- Semantic inconsistencies (keys, functional dependencies,. . .)