XML Data Exchange: Consistency and Query Answering

Marcelo Arenas  
U. of Toronto

Leonid Libkin  
U. of Toronto
The Problem of Data Exchange

- **Given:** A source schema $S$, a target schema $T$ and a specification $\Sigma$ of the relationship between these schemas.

- **Data exchange:** Problem of finding an instance of $T$, given an instance of $S$.
  
  - Target instance should reflect the source data as accurately as possible, given the constraints imposed by $\Sigma$ and $T$.
  
  - It should be efficiently computable.
  
  - It should allow one to evaluate queries on the target in a way that is semantically consistent with the source data.
Data Exchange

Source schema  Target schema
Data Exchange

Source database

Source schema

Target schema
Data Exchange

Source database -> Target database

Source schema  TARGET SCHEMA

Target database
Data Exchange

Source database

\[ \Sigma \]

Target database

Source schema

Target schema
Data Exchange

Query over the target: $Q$

Answer to $Q$ in the target instance should represent the answer to $Q$ in the space of possible translations of the source instance.
Data Exchange in Relational Databases

• Data exchange has been extensively studied in the relational world.
  - It has also been implemented: Clio.

• Relational data exchange settings:
  - Source and target schemas: Relational schemas.
  - Relationship between source and target schemas: Source-to-target dependencies.

• Semantics of data exchange has been precisely defined.
  - Algorithms for materializing target instances and for answering queries over the target have been developed.
Outline

- XML data exchange settings.
  - XML source-to-target dependencies.

- Consistency of XML data exchange settings.

- Query answering in XML data exchange.

- Final remarks.
Outline

- XML data exchange settings.
  - XML source-to-target dependencies.

- Consistency of XML data exchange settings.

- Query answering in XML data exchange.

- Final remarks.
XML Documents

xml

db

book

@title
“Algebra”

author

@name
“Hungerford”

@aff
“U. Washington”

book

@title
“Real Analysis”

author

@name
“Royden”

@aff
“Stanford”
XML Documents

DTD:

\[
db \rightarrow book^+
\]

\[
book \rightarrow author^+
\]

\[
author \rightarrow \varepsilon
\]
XML Documents

DTD:

\[
\begin{align*}
| & \quad db & \rightarrow & \quad book^+ \\
| & \quad book & \rightarrow & \quad author^+ \\
| & \quad author & \rightarrow & \quad \varepsilon \\
| & \quad book & \rightarrow & \quad @title \\
| & \quad author & \rightarrow & \quad @name, @aff
\end{align*}
\]
XML Data Exchange Settings

- Source and target schemas are given by DTDs.

- To specify the relationship between the source and the target schemas, we use source-to-target dependencies.

  To define these dependencies, we use tree patterns ...
Tree Patterns: Example

```
book
  @title x
  author
    @name y
```

The diagram represents a tree pattern with nodes for `book`, `@title`, `author`, and `@name`. The values `x` and `y` are associated with these nodes.
Tree Patterns: Example

book
@title $x$
@author y

book
@title “Algebra”
@author “Hungerford”
@aff “U. Washington”

$\ldots$
Tree Patterns: Example

```
book
  @title x
  author
    @name y

... db ...

book
  @title "Real Analysis"
  author
    @name "Royden"

    @aff "Stanford"
```
Collect tuples \((x, y)\): (Algebra, Hungerford), (Real Analysis, Royden)
Tree Patterns

- Tree patterns: XPath-like language.
  
  - Example: \( \text{book}(\text{@title} = x)[\text{author}(\text{@name} = y)] \)

- Language also includes wildcard \_ (matching more than one symbol) and descendant operator ///.
**XML Source-to-target Dependencies**

- **Source-to-target dependency (STD):**

  \[ \psi_T(x, z) \leftarrow \varphi_S(x, y), \]

  where \( \varphi_S(x, y) \) and \( \psi_T(x, z) \) are tree-pattern formulas over the source and target DTDs, resp.

- **Example:**

  - XML source:
    ```xml
    writer
     \langle name \rangle y
     \langle work \rangle
      \langle title \rangle x
      \langle year \rangle z
    
    book
     \langle title \rangle x
     \langle author \rangle
      \langle name \rangle y
    ```
XML Data Exchange Settings

XML Data Exchange Setting: \((D_S, D_T, \Sigma_{ST})\)

\(D_S\): Source DTD.

\(D_T\): Target DTD.

\(\Sigma_{ST}\): Set of XML source-to-target dependencies.

Each constraint in \(\Sigma_{ST}\) is of the form \(\psi_T(x, z) \leftarrow \varphi_S(x, y)\).

- \(\varphi_S(x, y)\): Tree-pattern formula over \(D_S\).
- \(\psi_T(x, z)\): Tree-pattern formula over \(D_T\).
XML Data Exchange Problem

- Given a source tree $T$, find a target tree $T'$ such that $(T, T')$ satisfies $\Sigma_{ST}$.

  - $(T, T')$ satisfies $\psi_T(x, z) := \varphi_S(x, y)$ if whenever $T$ satisfies $\varphi_S(\bar{a}, \bar{b})$, there is a tuple $\bar{c}$ such that $T'$ satisfies $\psi_T(\bar{a}, \bar{c})$.

  - $T'$ is called a solution for $T$. 

12
Example: Finding Solutions

Source  \[ db \rightarrow book^+ \]

DTD:

\[ book \rightarrow author^+ \quad \quad book \rightarrow \text{@title} \]
\[ author \rightarrow \varepsilon \quad \quad author \rightarrow \text{@name, @aff} \]

Target  \[ bib \rightarrow writer^+ \]

DTD:

\[ writer \rightarrow work^+ \quad \quad writer \rightarrow \text{@name} \]
\[ work \rightarrow \varepsilon \quad \quad work \rightarrow \text{@title, @year} \]

\[ \Sigma_{ST}: \]
writer
\[ \quad \text{@name} \quad work \]
\[ y \quad \text{@title} \quad \text{@year} \quad x \quad z \quad \text{@title} \quad \text{@year} \quad \text{@name} \quad y \]

\[ \vdash \]
Example: Finding Solutions

Let $T$ be our original tree:

```
db
    book
        @title
        "Algebra"
        @name
        "Hungerford"
        @aff
        "U. Washington"
    book
        @title
        "Real Analysis"
        @name
        "Royden"
        @aff
        "Stanford"
```
Example: Finding Solutions

A solution for $T$:

```
bib
  writer
    @name "Hungerford"
    work
      @title "Algebra"
      @year "1"
  writer
    @name "Royden"
    work
      @title "Real Analysis"
      @year "2"
```
Outline

- XML data exchange settings.
  - XML source-to-target dependencies.

- Consistency of XML data exchange settings.

- Query answering in XML data exchange.

- Final remarks.
Consistency of XML Data Exchange Settings

- An XML data exchange setting \((D_S, D_T, \Sigma_{ST})\) can be inconsistent:

  There are no \(T\) conforming to \(D_S\) and \(T'\) conforming to \(D_T\) such that \((T, T')\) satisfies \(\Sigma_{ST}\).

- What is the complexity of checking whether a setting is consistent?
Bad News: General Case

**Theorem** Checking if an XML data exchange setting is consistent is EXPTIME-complete.

Results on containment of XPath expressions as well as universality of tree automata imply that EXPTIME-hardness is unavoidable.
Good News: Consistency for Commonly used DTDs

A large number of DTDs that occur in practice have rules of the form:

\[ \ell \rightarrow \hat{\ell}_1, \ldots, \hat{\ell}_m, \]

where all the \( \ell_i \)'s are distinct, and \( \hat{\ell} \) is one of the following: \( \ell \), or \( \ell^* \), or \( \ell^+ \), or \( \ell? \)

Subsume non-relational data exchange handled by Clio.

**Theorem** For non-recursive DTDs that only have these rules, consistency can be checked in time \( O((\|D_S\| + \|D_T\|) \cdot \|\Sigma_{ST}\|^2) \).
Outline

- XML data exchange settings.
  - XML source-to-target dependencies.

- Consistency of XML data exchange settings.

- Query answering in XML data exchange.

- Final remarks.
• Decision to make: What is our query language?

• We start by considering a query language that produces tuples of values.
Conjunctive Tree Queries

- Query language $CTQ$ is defined by

$$ Q := \varphi \mid Q \land Q \mid \exists x Q, $$

where $\varphi$ ranges over tree-pattern formulas.

- By disallowing descendant $\now$ we obtain restriction $CTQ$. 
Example: Conjunctive Tree Query

List all pairs of authors that have written articles with the same title.

\[ Q(x, y) := \exists z \left( \begin{array}{c}
\text{name} \\
\text{work} \\
\text{title}
\end{array} \right)_x \land \left( \begin{array}{c}
\text{name} \\
\text{work} \\
\text{title}
\end{array} \right)_y \]
Certain Answers Semantics

- Given: A source tree $T$ and a conjunctive tree query $Q$ over the target.

- Answer to $Q$ should represent the answer to this query in the space of solutions for $T$.

- Certain answers semantics:

$$\text{certain}(Q, T') = \bigcap_{T' \text{ is a solution for } T} Q(T').$$
Computing Certain Answers

We study the following problem.

Given data exchange setting \((D_S, D_T, \Sigma_{ST})\) and query \(Q\):

**PROBLEM:** Certain-Answers\((Q)\).

**INPUT:** Tree \(T\) conforming to \(D_S\) and tuple \(\bar{a}\).

**QUESTION:** Is \(\bar{a} \in \text{certain}(Q, T)\)?
**Theorem** For every XML data exchange setting and $CTQ$-query $Q$, $\text{CERTAIN-ANSWERS}(Q)$ is in $\text{coNP}$.

Remark: In terms of the size of the document (data complexity).

**Theorem** There exist an XML data exchange setting and a $CTQ$-query $Q$ such that $\text{CERTAIN-ANSWERS}(Q)$ is $\text{coNP}$-hard.

We want to find tractable cases ...
**Theorem** Suppose one of the following is allowed in tree patterns over the target in STDs:

- descendant operator //, or
- wildcard _, or
- patterns that do not start at the root.

Then one can find source and target DTDs and a $CTQ$-query $Q$ such that $\text{CERTAIN-ANSWERS}(Q)$ is coNP-complete.

**Remark:** Even if all the rules in the DTDs are of the form:

$$\ell \rightarrow (\ell_1 | \cdots | \ell_n)^*$$

where all the $\ell_i$’s are distinct.
To find tractable cases, we have to concentrate on fully-specified STDs:

We impose restrictions on tree patterns over target DTDs:
- no descendant relation //; and
- no wildcard _; and
- all patterns start at the root.

No restrictions imposed on tree patterns over source DTDs.

- Subsume non-relational data exchange handled by Clio.

From now on, all STDs are fully-specified.
Computing Certain Answers: Towards a Classification

Given a class $C$ of regular expressions and a class $Q$ of queries:

$C$ is tractable for $Q$ if for every data exchange setting in which target DTDs only use regular expressions from $C$ and every $Q$-query $Q$, $CERTAIN-ANSWERS(Q)$ is in $PTIME$.

$C$ is coNP-complete for $Q$ if there is a data exchange setting in which target DTDs only use regular expressions from $C$ and a $Q$-query $Q$ such that $CERTAIN-ANSWERS(Q)$ is coNP-complete.

Remark (Ladner): If $PTIME \neq NP$, there are problems in coNP which are neither tractable nor coNP-complete.
Our classification is based on classes of regular expressions used in target DTDs.

We only impose one restriction to these classes: They must contain the simplest type of regular expressions.

Such classes will be called admissible.
Computing Certain Answers: Dichotomy

Theorem

1) Every admissible class $C$ of regular expressions is either tractable or coNP-complete for $CTQ//$. 

2) For every tractable class: Given a source tree $T$, one can compute in PTIME a solution $T^*$ for $T$ such that

$$\text{certain}(Q, T) = \text{remove_null_tuples}(Q(T^*)) .$$

3) It is decidable whether the regular expressions used in a target DTD belong to a tractable class.
A Tractable Class: Univocal Regular Expressions

- $C_U$: class of **univocal** regular expressions.
  - Non-univocal: $A, (B|C)$.

- Univocal regular expressions: Given a source tree $T$, one can compute in PTIME a solution $T^*$ for $T$ such that

$$\text{certain}(Q, T) = \text{remove_null_tuples}(Q(T^*))$$

- **Theorem** $C_U$ is tractable for $CTQ//$. 
Non-tractable Classes

Is there any other tractable class of regular expressions?

**Theorem** $C_U$ is maximal: If $C$ is an admissible class of regular expressions such that $C \not\subseteq C_U$, then $C$ is coNP-complete for $CTQ$-queries.

Dichotomy follows from this theorem and tractability of $C_U$.

**Theorem** It is decidable whether a regular expression is univocal.
Outline

- XML data exchange settings.
  - XML source-to-target dependencies.

- Consistency of XML data exchange settings.

- Query answering in XML data exchange.

- Final remarks.
Final Remarks

- Dichotomy also holds for unions of conjunctive queries.

- Future work:
  - We would like to consider XML query languages that produce XML trees.
    How do we define certain answers?
  - The notion of reasonable solutions needs to be investigated further.
Tractable Case: Univocal Regular Expressions

- $T^*$ is a canonical solution for $T$:

  $$\text{certain}(Q, T) = \text{remove\_null\_tuples}(Q(T^*))$$.

- We compute $T^*$ in two steps:
  
  - We use STDs to compute a canonical pre-solution $cps(T)$ from $T$.
  - Then we use target DTD to compute $T^*$ from $cps(T)$.
Example: XML Data Exchange Setting

- **Source DTD:**
  
  \[
  \begin{align*}
  r & \rightarrow A^*, B^* \\
  A & \rightarrow \varepsilon \quad \quad A & \rightarrow \@l \\
  B & \rightarrow \varepsilon \quad \quad B & \rightarrow \@l 
  \end{align*}
  \]

- **Target DTD:**
  
  \[
  \begin{align*}
  r & \rightarrow (C, D)^* \\
  C & \rightarrow \varepsilon \quad \quad C & \rightarrow \@m \\
  D & \rightarrow E \\
  E & \rightarrow \varepsilon \quad \quad E & \rightarrow \@n 
  \end{align*}
  \]

- **\( \Sigma_{ST} \):**

  \[
  \begin{align*}
  r[C(@m = x)] & \leftarrow A(@l = x), \\
  r[C(@m = x)] & \leftarrow B(@l = x). 
  \end{align*}
  \]
Example: Computing Canonical Pre-solution

```
   r
  / \
A   B
  |   |
@l  @l
“1” “2”
```
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution

\[
\begin{array}{c}
\text{\textit{Example: Computing Canonical Pre-solution}} \\
\end{array}
\]
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution

\[
\begin{align*}
  r & \downarrow \\
  C & \downarrow \\
  @m & \quad \text{“1”}
\end{align*}
\]

\[
\begin{align*}
  r & \quad A \\
  & \quad @\ell \\
  & \quad \text{“1”}
\end{align*}
\]

\[
\begin{align*}
  r & \quad B \\
  & \quad @\ell \\
  & \quad \text{“2”}
\end{align*}
\]
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution

\[ r \]
\[ C \]
\[ \@m \]
\[ “1” \]

\[ r \]
\[ C \]
\[ \@m \]
\[ “1” \]

\[ r \]
\[ C \]
\[ \@m \]
\[ x \]

\[ r \]
\[ C \]
\[ \@m \]
\[ x \]

\[ r \]
\[ A \]
\[ \@l \]
\[ “1” \]

\[ r \]
\[ B \]
\[ \@l \]
\[ “2” \]
Example: Computing Canonical Pre-solution

\[ \begin{array}{c}
 r \\
 \downarrow \\
 C \\
 \downarrow \\
 @m \\
 "1"
\end{array} \]

\[ \begin{array}{c}
 r \\
 \downarrow \\
 C \\
 \downarrow \\
 @m \\
 "2"
\end{array} \]

\[ \begin{array}{c}
 A \\
 \downarrow \\
 @l \\
 "1"
\end{array} \]

\[ \begin{array}{c}
 B \\
 \downarrow \\
 @l \\
 "2"
\end{array} \]

\[ \begin{array}{c}
 r \\
 \downarrow \\
 C \\
 \downarrow \\
 @m \\
 x
\end{array} \]

\[ \begin{array}{c}
 r \\
 \downarrow \\
 C \\
 \downarrow \\
 @m \\
 x
\end{array} \]

\[ \begin{array}{c}
 @l \\
 "x"
\end{array} \]

\[ \begin{array}{c}
 @l \\
 "x"
\end{array} \]
Example: Computing Canonical Pre-solution

\[ \begin{align*}
  r & \downarrow \\
  C & \downarrow \\
  @m & \uparrow \\
  \text{“1”} & \\
\end{align*} \]

\[ \begin{align*}
  r & \downarrow \\
  C & \downarrow \\
  @m & \uparrow \\
  \text{“2”} & \\
\end{align*} \]
Example: Computing Canonical Pre-solution

Canonical pre-solution:

Not yet a solution: It does not conform to the target DTD.
Example: Computing Canonical Solution

```
\begin{tikzpicture}
  \node (root) at (0,0) {$r$};
  \node (c1) at (-2,-2) {$C$};
  \node (c2) at (2,-2) {$C$};
  \draw[->] (root) -- (c1);
  \draw[->] (root) -- (c2);
  \node (m1) at (-3,-4) {$@m$ \text{“1”}};
  \node (m2) at (3,-4) {$@m$ \text{“2”}};
  \draw[->] (c1) -- (m1);
  \draw[->] (c2) -- (m2);
\end{tikzpicture}
```
Example: Computing Canonical Solution

\[ r \rightarrow (C, D)^* \]
Example: Computing Canonical Solution

\[ r \rightarrow (C, D)^* \]
Example: Computing Canonical Solution

\[ D \rightarrow E \]
Example: Computing Canonical Solution

\[ D \rightarrow E \]
Example: Computing Canonical Solution

\[
\begin{align*}
r & \\
C & \rightarrow \quad D & \rightarrow \quad C \\
\text{@}m \quad \text{“1”} & \quad E & \quad \text{@}m \quad \text{“2”} \\
\end{align*}
\]

\[E \rightarrow \text{@}n\]
Example: Computing Canonical Solution

\[ E \rightarrow @n \]
Example: Computing Canonical Solution

\[
\begin{array}{c}
\text{Example: Computing Canonical Solution} \\
\begin{tikzpicture}
  \node (r) {$r$};
  \node (C) [below left of=r] {$C$};
  \node (D) [below right of=r] {$D$};
  \node (C2) [below right of=D] {$C$};
  \node (D2) [below right of=C2] {$D$};
  \node (m1) [below of=C] {$@m$ \text{“1”}};
  \node (E) [below of=D] {$E$};
  \node (m2) [below of=C2] {$@m$ \text{“2”}};
  \node (n1) [below of=E] {$@n$ \text{“1”}};

  \draw[->] (r) -- (C);
  \draw[->] (r) -- (D);
  \draw[->] (D) -- (E);
  \draw[->] (C2) -- (m2);
  \draw[->] (E) -- (n1);
\end{tikzpicture}
\end{array}
\]
Example: Computing Canonical Solution

\[ D \rightarrow E \]
Example: Computing Canonical Solution

\[
E \rightarrow \@n
\]
Example: Computing Canonical Solution

\[ E \rightarrow @n \]
Example: Computing Canonical Solution

\[ r \]

\[ C \]
\[ @m \]
\[ "1" \]
\[ @n \]
\[ "\bot_1" \]

\[ D \]
\[ E \]
\[ @m \]
\[ "2" \]
\[ @n \]
\[ "\bot_2" \]

\[ C \]
\[ D \]