XML Data Exchange: Consistency and Query Answering

Marcelo Arenas
U. of Toronto

Leonid Libkin
U. of Toronto
The Problem of Data Exchange

- Data exchange is the problem of finding an instance of a target schema, given an instance of a source schema and a specification of the relationship between the source and the target.

- Such a target instance should correctly represent information from the source instance under the constraints imposed by the target schema.

- It should also allow one to evaluate queries on the target instance in a way that is semantically consistent with the source data.
Data Exchange

Source schema \hspace{1cm} Target schema
Data Exchange

Source Database

Target Database

Source schema

Target schema
Query over the target schema: $Q$

How to answer $Q$ so that the answer is consistent with the data in the source database?
Relational Data Exchange Settings

Data Exchange Setting: \((S, T, \Sigma_{ST})\)

- **S**: Source schema.
- **T**: Target schema.
- **\(\Sigma_{ST}\)**: Set of source-to-target dependencies.
  - Source-to-target dependency:
    \[
    \psi_T(\bar{x}, \bar{z}) := \varphi_S(\bar{x}, \bar{y}).
    \]
    - \(\varphi_S(\bar{x}, \bar{y})\): conjunction of atomic formulas over \(S\).
    - \(\psi_T(\bar{x}, \bar{z})\): conjunction of atomic formulas over \(T\).
Example: Relational Data Exchange Setting

- \( S = Book(Title, AName, Aff) \)

- \( T = Writer(Name, BTitle, Year) \)

- \( \Sigma_{ST} = Writer(x_2, x_1, z_1) \unicode{x2970} Book(x_1, x_2, y_1) \).
Relational Data Exchange Problem

- Given a source instance $I$, find a target instance $J$ such that $(I, J)$ satisfies $\Sigma_{ST}$.
  
  - $(I, J)$ satisfies $\psi_T(\bar{x}, \bar{z}) := \varphi_S(\bar{x}, \bar{y})$ if whenever $I$ satisfies $\varphi_S(\bar{a}, \bar{b})$, there is a tuple $\bar{c}$ such that $J$ satisfies $\psi_T(\bar{a}, \bar{c})$.
  
  - $J$ is called a solution for $I$.

- Previous example:

<table>
<thead>
<tr>
<th>Book</th>
<th>Title</th>
<th>AName</th>
<th>Aff</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I$:</td>
<td>Algebra</td>
<td>Hungerford</td>
<td>U. Washington</td>
</tr>
<tr>
<td>Real Analysis</td>
<td>Royden</td>
<td></td>
<td>Stanford</td>
</tr>
</tbody>
</table>
Relational Data Exchange Problem

Possible solutions:

<table>
<thead>
<tr>
<th>Writer</th>
<th>Name</th>
<th>BTitle</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hungerford</td>
<td>Algebra</td>
<td>1974</td>
</tr>
<tr>
<td></td>
<td>Royden</td>
<td>Real Analysis</td>
<td>1988</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Writer</th>
<th>Name</th>
<th>BTitle</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hungerford</td>
<td>Algebra</td>
<td>$\perp_1$</td>
</tr>
<tr>
<td></td>
<td>Royden</td>
<td>Real Analysis</td>
<td>$\perp_2$</td>
</tr>
</tbody>
</table>
Query Answering

• $Q$ is a query over target schema.

What does it mean to answer $Q$?

$$\text{certain}(Q, I) = \bigcap_{J \text{ is a solution for } I} Q(J)$$

• Previous example:

- $$\text{certain}(\exists y \exists z \text{ Writer}(x, y, z), I) = \{\text{Hungerford, Royden}\}$$
Outline

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

• Consistency of XML data exchange settings.

• Query answering in XML data exchange.

• Future work.
XML Documents

```
<db>
  <book>
    <title>Algebra</title>
    <author>
      <name>Hungerford</name>
      <affiliation>U. Washington</affiliation>
    </author>
  </book>
  <book>
    <title>Real Analysis</title>
    <author>
      <name>Royden</name>
      <affiliation>Stanford</affiliation>
    </author>
  </book>
</db>
```

**DTD:**

```
db  →  book+
book →  author+
author →  ε
```
XML Documents

\[
db 
\downarrow
\text{book}
\downarrow
\text{author}
\quad \text{title} \quad \text{name}, \text{aff}
\quad \text{“Algebra”} \quad \text{“Hungerford” “U. Washington”}
\quad \text{“Real Analysis”} \quad \text{“Royden” “Stanford”}
\]

\[
\text{DTD:}
\quad \text{db} \rightarrow \text{book}^+
\quad \text{book} \rightarrow \text{author}^+
\quad \text{author} \rightarrow \varepsilon
\quad \text{book} \rightarrow \text{@title}
\quad \text{author} \rightarrow \text{@name, @aff}
\]
Instead of source and target relational schemas, we have source and target DTDs.

But what are the source-to-target dependencies?

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

```
book
  @title x
  author
    @name y
```
Tree Patterns: Example

```
book
  @title x
  @name y

author
```

```
book
  @title “Algebra”
  @name “Hungerford”

author
  @aff “U. Washington”
```

$db$

...
Tree Patterns: Example

\[
\begin{array}{c}
\text{book} \\
\text{@title } x \\
\text{author} \\
\text{@name } y \\
\end{array}
\]

\[
\begin{array}{c}
\text{db} \\
\text{book} \\
\text{@title } \text{“Real Analysis”} \\
\text{author} \\
\text{@name } \text{“Royden”} \\
\text{@aff } \text{“Stanford”} \\
\end{array}
\]
Tree Patterns: Example

Collect tuples $(x, y)$: (Algebra, Hungerford), (Real Analysis, Royden)
Tree Patterns

- Example: $book(\text{title} = x)[author(\text{name} = y)]$.

- 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 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 \(\psi_T(\bar{x}, \bar{z}) : \neg \varphi_S(\bar{x}, \bar{y})\).

- \(\varphi_S(\bar{x}, \bar{y})\): tree-pattern formula over \(D_S\).
- \(\psi_T(\bar{x}, \bar{z})\): tree-pattern formula over \(D_T\).
Example: XML Data Exchange Setting

- **Source DTD:**
  
  \[
  \begin{align*}
  db & \rightarrow book^+ \\
  book & \rightarrow author^+ \\
  author & \rightarrow \varepsilon
  \end{align*}
  \]

- **Target DTD:**
  
  \[
  \begin{align*}
  bib & \rightarrow writer^+ \\
  writer & \rightarrow work^+ \\
  work & \rightarrow \varepsilon
  \end{align*}
  \]

- **\( \Sigma_{ST} \):**
  
  \[
  writer(\@name = y)[work(\@title = x, \@year = z)] \leftarrow \\
  book(\@title = x)[author(\@name = y)].
  \]
XML Data Exchange Problem

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

  - $(T, T')$ satisfies $\psi_T(\bar{x}, \bar{z}) := \varphi_S(\bar{x}, \bar{y})$ 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 $T$. 
Let $T$ be our original tree:
XML Data Exchange Problem

A solution for $T$:

```
          bib
         /   \
writer  writer
 /      /   \
@name  @name
  "Hungerford" "Royden"
   /  /   \
@title @title
  "Algebra" "Real Analysis"
   /  /    \
@year @year
  "1974" "1988"
```

@year 1988
XML Data Exchange Problem

Another solution for $T$:

```
bib
  ├── writer
  │   └── work
  │       ├── @name: "Hungerford"
  │       │   └── @title: "Algebra"
  │       └── @year: "19"
  └── writer
        └── work
             ├── @name: "Royden"
             │   └── @title: "Real Analysis"
             └── @year: "2012"
```
Consistency of XML Data Exchange Settings

• What if we have target DTD

\[
\begin{align*}
\text{bib} & \rightarrow \text{writer}^+ \\
\text{writer} & \rightarrow \text{novelist}^*, \text{poet}^* \\
\text{novelist} & \rightarrow \text{work}^+ \\
\text{poet} & \rightarrow \text{work}^+ \\
\text{work} & \rightarrow \varepsilon
\end{align*}
\]

writer \rightarrow @name

work \rightarrow @title, @year

in our previous example?

• The setting becomes inconsistent!

- There are no T conforming to $D_S$ and $T'$ conforming to $D_T$ such that $(T, T')$ satisfies $\Sigma_{ST}$. 
Consistency of XML Data Exchange Settings

- An XML data exchange setting is **inconsistent** if it does not admit solutions for any given source tree. Otherwise it is **consistent**.

- A relational data exchange setting is always consistent.

- An XML data exchange setting is not always consistent.
  - 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.

Known results on containment of XPath expressions as well as universality of tree automata imply that EXPTIME-hardness is unavoidable.
A large number of DTDs that occur in practice have rules of the following 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 \( \ell? \)

**Theorem** For non-recursive DTDs that only have these rules, checking if an XML data exchange setting is consistent is solvable in time \( O((\|D_S\| + \|D_T\|) \cdot \|\Sigma_{ST}\|^2)). \)
Query Answering in XML Data Exchange

- Decision to make: what is our query language?

- XML query languages such as XQuery take XML trees and produce XML trees.
  - This makes it hard to talk about certain answers.

- We use 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 $//$, we obtain restriction $CTQ$.

• Results extend to unions of conjunctive queries.
Example: Conjunctive Tree Query

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

\[ Q(x, y) := \exists z \left( \text{writer} \left( \text{@name} x, \text{work} \left( \text{@title} z \right), \text{writer} \left( \text{@name} y, \text{work} \left( \text{@title} z \right) \right) \right) \land \neg \right) \]
Computing Certain Answers

- Semantics: as in the relational case.

\[
certain(Q, T) = \bigcap_{T' \text{ is a solution for } T} Q(T').
\]

- Given data exchange setting \((D_S, D_T, \Sigma_{ST})\) and query \(Q\):
  
  PROBLEM: \text{CERTAnsw}(Q).
  
  INPUT: Tree \(T\) conforming to \(D_S\) and tuple \(\bar{a}\).
  
  QUESTION: Is \(\bar{a} \in certain(Q, T)\)?
Theorem For every XML data exchange setting and $CTQ$-query $Q$, $\text{CERTAnsw}(Q)$ 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 $\text{CERTAnsw}(Q)$ 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 $CERT\textsc{Answ}(Q)$ 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.
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 $Q$, $\text{CERTANSW}(Q)$ 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 $\text{CERTANSW}(Q)$ is coNP-complete.

Remark (Ladner): if PTIME $\neq$ NP, there are problems in coNP which are neither tractable nor coNP-complete.
Computing Certain Answers: Towards a Classification

• Our classification is based on classes of regular expressions used in DTDs.

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

• Such classes will be called admissible.
Theorem

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

Remark: also holds for unions of conjunctive queries.

2) Moreover, given an XML data exchange setting, it is decidable whether the regular expressions used in the source and the target DTD belong to a tractable class.
A Tractable Case

• Idea: given a source tree $T$, compute a solution $T^*$ for $T$ such that

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

• $T^*$ is a canonical solution for $T$.

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

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

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

\[
\begin{align*}
   r & \quad \downarrow \\
   C & \quad \downarrow \quad A \\
   @m & \quad @l \\
   x & \quad x
\end{align*}
\]

\[
\begin{align*}
   r & \quad \downarrow \\
   A & \quad \downarrow \\
   @l & \quad @l \\
   \text{“1”} & \quad \text{“2”}
\end{align*}
\]
Example: Computing Canonical Pre-solution

\[
C^r \quad : \quad A^r
\]

\[
C \quad \downarrow \quad \downarrow \quad A
\]

\[
\@m \quad \@l
\]

\[
x \quad x
\]

\[
A \quad B
\]

\[
\@l \quad \@l
\]

\[
“1” \quad “2”
\]
Example: Computing Canonical Pre-solution

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

\[
\begin{array}{c}
\uparrow r \\
\downarrow \\
\uparrow C \\
\downarrow \\
\uparrow @m \\
\text{“1”}
\end{array}
\]
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution

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

\[
\begin{align*}
  r & \downarrow \\
  C & \downarrow \\
  @m & \text{x} \\
  x & \downarrow \\
  A & \downarrow \\
  @l & \text{“1”} \\
  B & \downarrow \\
  @l & \text{“2”}
\end{align*}
\]
Example: Computing Canonical Pre-solution

\[ \begin{array}{c}
\text{r} \\
\downarrow \\
\text{C} \\
\downarrow \\
@m \\
\text{“1”}
\end{array} \quad \begin{array}{c}
\text{r} \\
\downarrow \\
\text{C} \\
\downarrow \\
@m \\
x \\
@l \\
x
\end{array} \quad \begin{array}{c}
\text{r} \\
\downarrow \\
\text{A} \\
@l \\
\text{“1”}
\end{array} \quad \begin{array}{c}
\text{r} \\
\downarrow \\
\text{B} \\
@l \\
\text{“2”}
\end{array} \]
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution

Canonical pre-solution:

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

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

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

\[
\begin{array}{c}
\text{Example: Computing Canonical Solution} \\
\end{array}
\]

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

\[ \text{Example: Computing Canonical Solution} \]

![Diagram](image-url)

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

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

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

\[
\begin{align*}
\mathcal{D} & \rightarrow E \\
C & \rightarrow D \\
\text{@}m & \rightarrow E \\
\text{“1”} & \\
\mathcal{D} & \rightarrow \text{C} \\
\text{@}m & \rightarrow \text{C} \\
\text{“2”} & \\
\cdots & \\
\text{D} & \rightarrow \text{E} \\
\end{align*}
\]
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
A Tractable Case: Univocal Regular Expressions

- $\mathcal{C}_U$: class of **univocal** regular expressions.
  - Non-univocal: $A, (B|C)$.

- For target DTDs only using univocal regular expressions:
  - There exists a solution for a tree $T$ iff there exists a canonical solution $T^*$ for $T$.
  - Previous algorithm computes canonical solution $T^*$ for $T$ in polynomial time.
  - $\text{certain}(Q, T) = \text{remove\_null\_tuples}(Q(T^*))$, for every $\text{CTQ}//$ query.

- **Theorem** $\mathcal{C}_U$ is tractable for $\text{CTQ}//$.
Is there any other tractable class of regular expressions?

**Theorem** \( C_U \) is the maximal tractable class: 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.
Future Work

- What about XML languages like XQuery that return XML documents? How do we define certain answers?

- The notion of reasonable solutions needs to be investigated further.

We would like to consider different certain-answers semantics.