XML Data Exchange
Relational Data Exchange Settings

Data Exchange Setting: \((\sigma, \tau, \Sigma)\)

\(\sigma\): Source schema.

\(\tau\): Target schema.

\(\Sigma\): Set of rules that specify relationship between the target and the source (source-to-target dependencies).
  - Source-to-target dependency:
    \[
    \psi_\tau(\bar{x}, \bar{z}) \leftarrow \varphi_\sigma(\bar{x}, \bar{y}).
    \]
  - \(\varphi_\sigma(\bar{x}, \bar{y})\): conjunction of atomic formulas over \(\sigma\).
  - \(\psi_\tau(\bar{x}, \bar{z})\): conjunction of atomic formulas over \(\tau\).
Example: Relational Data Exchange Setting

\[
\begin{align*}
\sigma &= Book(Title, AName, Aff) \\
\tau &= Writer(Name, BTitle, Year) \\
\Sigma &= Writer(x_2, x_1, z_1) \leftarrow Book(x_1, x_2, y_1).
\end{align*}
\]
Relational Data Exchange Problem

- Given a source instance $S$, find a target instance $T$ such that $(S, T)$ satisfies $\Sigma$.

  - $(S, T)$ satisfies $\psi_{\tau}(\bar{x}, \bar{z}) \leftarrow \varphi_{\sigma}(\bar{x}, \bar{y})$ if whenever $S$ satisfies $\varphi_{\sigma}(\bar{a}, \bar{b})$, there is a tuple $\bar{c}$ such that $T$ satisfies $\psi_{\tau}(\bar{a}, \bar{c})$.

  - $T$ is called a solution for $S$.

- Previous example:

<table>
<thead>
<tr>
<th>Book</th>
<th>Title</th>
<th>AName</th>
<th>Aff</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$:</td>
<td>Algebra</td>
<td>Hungerford</td>
<td>U. Washington</td>
</tr>
<tr>
<td>Real Analysis</td>
<td>Royden</td>
<td>Stanford</td>
<td></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>T1:</td>
<td></td>
<td></td>
<td></td>
</tr>
<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>T2:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hungerford</td>
<td>Algebra</td>
<td>⊥₁</td>
</tr>
<tr>
<td></td>
<td>Royden</td>
<td>Real Analysis</td>
<td>⊥₂</td>
</tr>
</tbody>
</table>
Query Answering

- \( Q \) is a query over target schema.

What does it mean to answer \( Q \)?

\[
certain(Q, S) = \bigcap_{T \text{ is a solution for } S} \bigcap_{R \in \text{POSS}(T)} Q(R)
\]

- Previous example:

\[
certain(\exists y \exists z \text{ Writer}(x, y, z), I) = \{\text{Hungerford, Royden}\}
\]
XML Documents

DTD:

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

```
 db
 /
/  
book --- book
|   |   |
@title @name  @name | @title
Algebra "Hungerford" "U. Washington"

 mutually recursive

DTD:
  db -> book+
  book -> author+
  author -> e
  book -> @title
  author -> @name, @aff
```
XML Data Exchange Settings

• 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.

  If a certain pattern is found in the source, another pattern has to be found in the target.
Tree Patterns: Example
Tree Patterns: Example

```
  db
     /--
     |
  book
     /--
     |
@title  x
     /--
     |
author
     /--
     |
@name  y

  book
     /--
     |
@title  "Algebra"
     /--
     |
author
     /--
     |
@name  "Hungerford"
     /--
     |
@aff   "U. Washington"
```

Tree Patterns: Example

\[ \text{db} \]

\[ \text{book} \]

\[ \text{@title} \ x \]

\[ \text{@name} \ y \]

\[ \ldots \]

\[ \text{book} \]

\[ \text{@title} \ “Real Analysis” \]

\[ \text{@name} \ “Royden” \]

\[ \text{@aff} \ “Stanford” \]
Collect tuples \((x, y):\) (Algebra, Hungerford), (Real Analysis, Royden)
Tree Patterns

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

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

- Source-to-target dependency (STD):

\[ \psi_T(\bar{x}, \bar{z}) := \varphi_\sigma(\bar{x}, \bar{y}), \]

where \( \varphi_\sigma(\bar{x}, \bar{y}) \) and \( \psi_T(\bar{x}, \bar{z}) \) are tree-patterns over the source and target DTDs, resp.

- Example:
XML Data Exchange Settings

XML Data Exchange Setting: \((D_\sigma, D_\tau, \Sigma)\)

\(D_\sigma\): Source DTD.

\(D_\tau\): Target DTD.

\(\Sigma\): Set of XML source-to-target dependencies.

Each constraint in \(\Sigma\) is of the form \(\psi_\tau(\bar{x}, \bar{z}) :\neg \varphi_\sigma(\bar{x}, \bar{y})\).

- \(\varphi_\sigma(\bar{x}, \bar{y})\): tree-pattern over \(D_\sigma\).
- \(\psi_\tau(\bar{x}, \bar{z})\): tree-pattern over \(D_\tau\).
Example: XML Data Exchange Setting

- **Source DTD:**
  
  \[
  \begin{align*}
  db & \rightarrow \text{book}^+ \\
  \text{book} & \rightarrow \text{author}^+ & \text{book} & \rightarrow @\text{title} \\
  \text{author} & \rightarrow \varepsilon & \text{author} & \rightarrow @\text{name}, @\text{aff} \\
  \end{align*}
  \]

- **Target DTD:**
  
  \[
  \begin{align*}
  \text{bib} & \rightarrow \text{writer}^+ \\
  \text{writer} & \rightarrow \text{work}^+ & \text{writer} & \rightarrow @\text{name} \\
  \text{work} & \rightarrow \varepsilon & \text{work} & \rightarrow @\text{title}, @\text{year} \\
  \end{align*}
  \]

- \(\Sigma:\)

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

- Given a source tree $T$, find a target tree $T'$ such that $(T, T')$ satisfies $\Sigma$.

  - $(T, T')$ satisfies $\psi_\tau(\bar{x}, \bar{z}) :\neg \varphi_\sigma(\bar{x}, \bar{y})$ if whenever $T$ satisfies $\varphi_\sigma(\bar{a}, \bar{b})$, there is a tuple $\bar{c}$ such that $T'$ satisfies $\psi_\tau(\bar{a}, \bar{c})$.

  - $T'$ is called a solution for $T$. 
XML Data Exchange Problem

Let $T$ be our original tree:

```
<db>
  <book>
    <@title>“Algebra”</@title>
    <author>
      <@name>“Hungerford”</@name>
      <@aff>“U. Washington”</@aff>
    </author>
  </book>
  <book>
    <@title>“Real Analysis”</@title>
    <author>
      <@name>“Royden”</@name>
      <@aff>“Stanford”</@aff>
    </author>
  </book>
</db>
```
XML Data Exchange Problem

A solution for $T$:
XML Data Exchange Problem

Another solution for $T$:

```
<bib>
  <writer>
    <name>“Hungerford”</name>
    <work>
      <title>“Algebra”</title>
      <year>1</year>
    </work>
  </writer>
  <writer>
    <name>“Royden”</name>
    <work>
      <title>“Real Analysis”</title>
      <year>2</year>
    </work>
  </writer>
</bib>
```
Consistency of XML Data Exchange Settings

• What if we have target DTD

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

in our previous example?

• The setting becomes inconsistent!

- There are no $T$ conforming to $D_\sigma$ and $T'$ conforming to $D_\tau$ such that $(T,T')$ satisfies $\Sigma$.
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

Fact Checking if an XML data exchange setting is consistent necessarily takes exponential time.

Complexity-theoretic statement: \text{EXPTIME-complete}.

But the parameter is the size of the DTDs and constraints – typically not very large. Hence $2^{O(n)}$ is not too bad.
Good News: Consistency for Commonly used DTDs

DTDs that commonly occur in practice tend to be simple. In fact more than 50% of regular expressions are of this form:

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

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

For example, \( \text{book} \rightarrow \text{title, author}^+, \text{chapter}^*, \text{publisher}? \)

A better algorithm For non-recursive DTDs that only have these rules, checking if an XML data exchange setting is consistent is solvable in time \( O((\|D_\sigma\| + \|D_\tau\|) \cdot \|\Sigma\|^2) \).
• 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.

• For now we use a query language that produces tuples of values.
Conjunctive Tree Queries

- Query language $\text{CTQ}$ is defined by

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

where $\varphi$ ranges over tree-patterns.

- Reminder: relational conjunctive queries are defined by the same rules where $\varphi$ ranges over relational atoms (i.e., formulas $R(x_1, \ldots, x_n)$).
Example: Conjunctive Tree Query

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

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

• Semantics: as in the relational case.

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

• Given data exchange setting \((D_\sigma, D_\tau, \Sigma)\) and query \(Q\):

• **PROBLEM:** For a tree \(T\) conforming to \(D_\sigma\), compute \(\text{\textit{certain}}(Q, T)\)
Computing Certain Answers: General Picture

It is not even clear if the problem is solvable.

**Good news** For every XML data exchange setting and $CTQ$-query $Q$, the problem $CERTANSW(Q)$ is solvable in exponential time.

**Not so good news** Sometimes exponential time is unavoidable (the problem ma be coNP-complete)

We want to find cases that admit fast algorithms.
Computing Certain Answers: Eliminating bad cases

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 (in fact, very simple DTDs) and a $\mathcal{CTQ}$-query $Q$ such that $\text{CERTANSW}(Q)$ must take exponential time.

A more precise statement: is coNP-complete.
Fully specified constraints

- We disallow the three features that make query answering hard.

- This gives us 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 IBM.
An efficient case

- Recall relational data exchange and conjunctive queries: then \(\text{certain}(Q, S) = \text{certain}(Q, \text{CanSol}(S))\).

- **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:
  \[
  r \rightarrow A^*, B^* \\
  A \rightarrow \varepsilon \quad A \rightarrow @\ell \\
  B \rightarrow \varepsilon \quad B \rightarrow @\ell 
  \]

- Target DTD:
  \[
  r \rightarrow (C, D)^* \\
  C \rightarrow \varepsilon \quad C \rightarrow @m \\
  D \rightarrow E \\
  E \rightarrow \varepsilon \quad E \rightarrow @n 
  \]

- \( \Sigma \):
  \[
  r[C(@m = x)] :\equiv A(@\ell = x), \\
  r[C(@m = x)] :\equiv B(@\ell = x). 
  \]
Example: Computing Canonical Pre-solution

\[
\begin{array}{c}
\text{\texttt{r}} \\
\text{\texttt{A}} & \text{\texttt{B}} \\
\text{\texttt{1}} & \text{\texttt{2}} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\texttt{r}} \\
\text{\texttt{A}} & \text{\texttt{B}} \\
\text{\texttt{1}} & \text{\texttt{2}} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\texttt{r}} \\
\text{\texttt{A}} & \text{\texttt{B}} \\
\text{\texttt{1}} & \text{\texttt{2}} \\
\end{array}
\]
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution

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

\[ r \rightarrow C \rightarrow \text{@m} \rightarrow \text{"1"} \]
Example: Computing Canonical Pre-solution

\[ \begin{array}{c}
    r \\
    \downarrow \\
    C \\
    \downarrow \\
    \text{atm} \\
    "1"
\end{array} \]
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution

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

Canonical pre-solution:

```
   r
  / \  \
 C   C
  / \  \
 @m@m
 “1” “2”
```

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

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

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

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

\[ r \rightarrow \text{@n} \]

\[ C \rightarrow \text{@m} \text{“1”} \]
\[ D \rightarrow E \]
\[ C \rightarrow \text{@m} \text{“2”} \]
\[ D \]

\[ E \rightarrow \text{@n} \text{“⊥₁”} \]
Example: Computing Canonical Solution

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

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

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

\[ E \rightarrow @n \]
Example: Computing Canonical Solution
Does this always work?

Depends on regular expressions in target DTDs.

- class of good regular expressions.

  - Examples: \((A|B)^*\), \(A, B^+, C^*, D?\), \((A^*|B^*)\), \((C, D)^*\).
  - bad: \(A, (B|C')\).
  - exact definition: quite involved.
Does this always work? cont’d

- For target DTDs only using good 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 CTQ// query.

- Complexity: polynomial time.