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

- $\sigma = \text{Book}(\text{Title}, \text{AName}, \text{Aff})$

- $\tau = \text{Writer}(\text{Name}, \text{BTitle}, \text{Year})$

- $\Sigma = \text{Writer}(x_2, x_1, z_1) : \text{Book}(x_1, x_2, y_1)$. 
Relational Data Exchange Problem

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

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

  - $T$ 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></td>
<td>Real Analysis</td>
<td>Royden</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 $Q$?

$$\text{certain}(Q, S) = \bigcap \bigcap Q(R)$$

$T$ is a solution for $S$, $R \in \text{POSS}(T)$

• Previous example:

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

\[
\begin{align*}
&\text{DTD :} \\
&db \rightarrow book^+ \\
&book \rightarrow author^+ \\
&author \rightarrow \varepsilon
\end{align*}
\]
XML Documents

DTD:

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

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

```
book
  @title x
  author
    @name y

book
  @title "Algebra"
  author
    @name "Hungerford"
    @aff "U. Washington"

db
  ...
```
Tree Patterns: Example

\[
\text{\texttt{db}}
\]

\[
\text{\texttt{book}}
\]

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

\[
\text{\texttt{author}}
\]

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

\[
\ldots
\]

\[
\text{\texttt{book}}
\]

\[
\text{\texttt{@title}} \quad \text{\texttt{“Real Analysis”}}
\]

\[
\text{\texttt{author}}
\]

\[
\text{\texttt{@name}} \quad \text{\texttt{“Royden”}}
\]

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

\[
\psi_{\tau}(\bar{x}, \bar{z}) :\neg \varphi_{\sigma}(\bar{x}, \bar{y}),
\]

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

• Example:

```
writer
@name y
work
@title x
@year z
:=

book
@title x
author
@name y
```
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 \(\psi_\tau(\bar{x}, \bar{z}) : - \varphi_\sigma(\bar{x}, \bar{y}).\)

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

- Source DTD:

  \[
  \begin{align*}
  db & \rightarrow \text{book}^+ \\
  \text{book} & \rightarrow \text{author}^+ & \text{book} & \rightarrow @\text{title} \\
  \text{author} & \rightarrow \epsilon & \text{author} & \rightarrow @\text{name}, @\text{aff}
  \end{align*}
  \]

- Target DTD:

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

- \( \Sigma \):

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

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

  - $(T, T')$ satisfies $\psi_\tau(\bar{x}, \bar{z})$ if whenever $T$ satisfies $\varphi_\sigma(\bar{x}, \bar{y})$, 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>
      <affiliation>U. Washington</affiliation>
    </author>
  </book>
  <book>
    <title>Real Analysis</title>
    <author>
      <name>Royden</name>
      <affiliation>Stanford</affiliation>
    </author>
  </book>
</db>
```
XML Data Exchange Problem

A solution for $T$:

```
bib
  writer
    @name "Hungerford"
    work
      @title "Algebra"
      @year "1974"
  writer
    @name "Royden"
    work
      @title "Real Analysis"
      @year "1988"
```
Another solution for $T$:

```
<bib>
  <writer name="Hungerford">
    <work title="Algebra" @year="\bot_1"/>
  </writer>
  <writer name="Royden">
    <work title="Real Analysis" @year="\bot_2"/>
  </writer>
</bib>
```
Consistency of XML Data Exchange Settings

• What if we have target DTD

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

writer → @name

work → @title, @year

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

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


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:

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

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

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

**Theorem** For non-recursive DTDs that only have these rules, checking if an XML data exchange setting is consistent is solvable in time \( O((\|D_\sigma\| + \|D_\tau\|) \cdot \|\Sigma\|^2) \).
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 \( CTQ// \) is defined by

\[
Q ::= \varphi \mid Q \land Q \mid \exists x \ Q,
\]

where \( \varphi \) ranges over tree-patterns.

• By disallowing descendant \( // \) we obtain restriction \( CTQ \).
Example: Conjunctive Tree Query

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

\[ Q(x, y) := \]

\[ \exists z \ ( \text{writer} \ (\text{name} \ x \ \text{work} \ \text{title} \ z) \land \text{writer} \ (\text{name} \ y \ \text{work} \ \text{title} \ z) ) \]
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 \((D_\sigma, D_\tau, \Sigma)\) and query \(Q\):

PROBLEM: \(\text{CERT\textsc{AnsW}}(Q)\).

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

QUESTION: Is \(\bar{a} \in certain(Q,T)\)?
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 $\text{CERTAnsw}(Q)$ is solvable in exponential time.

More precisely, it is in $\text{coNP}$.

**Not so good news** Sometimes exponential time is “unavoidable”: There exist an XML data exchange setting and a $CTQ$-query $Q$ such that $\text{CERTAnsw}(Q)$ is $\text{coNP}$-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 $CTQ$-query $Q$ such that $\text{CERTANSW}(Q)$ must take exponential time.

A more precise statement: is $\text{coNP}$-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:

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

• \( \Sigma \):

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

```
\begin{tikzpicture}

  \node (C) at (0,0) {$C$};
  \node (A) at (1,0) {$A$};
  \node (m) at (0,-1) {$@m$};
  \node (l) at (1,-1) {$@l$};

  \draw[->] (C) -- (m);
  \draw[->] (C) -- (A);
  \draw[->] (A) -- (l);

  \node (x) at (0,-2) {$x$};
  \node (y) at (1,-2) {$x$};

  \node (r) at (2,0) {$r$};
  \node (A) at (3,0) {$A$};
  \node (B) at (4,0) {$B$};
  \node (l) at (3,-1) {$@l$};
  \node (m) at (4,-1) {$@l$};

  \node (1) at (3,-2) {"1"};
  \node (2) at (4,-2) {"2"};

  \draw[->] (r) -- (A);
  \draw[->] (r) -- (B);
  \draw[->] (A) -- (l);
  \draw[->] (B) -- (m);

\end{tikzpicture}
```
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution

\[ \begin{align*}
  r & \Downarrow C \\
  & \Downarrow @m \text{ "1"} \\
  & \Downarrow x
\end{align*} \quad \begin{align*}
  r & \Downarrow C \\
  & \Downarrow @m \text{ "1"} \\
  & \Downarrow x
\end{align*} \quad \begin{align*}
  r & \\
  & \Downarrow @m \text{ "1"} \\
  & \Downarrow @l \text{ "2"}
\end{align*} \quad \begin{align*}
  r & \\
  & \Downarrow @l \text{ "2"}
\end{align*} \quad \begin{align*}
  r & \\
  & \Downarrow @l \text{ "2"}
\end{align*} \]
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution
Example: Computing Canonical Pre-solution

\[ r \]
\[ \downarrow \]
\[ C \]
\[ \downarrow \]
\[ \@m \]
\[ “1” \]

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

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

\[ A \]
\[ \downarrow \]
\[ \@l \]
\[ “1” \]

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

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

\[
\begin{array}{c}
\begin{array}{c}
  r \\
  \downarrow \\
  C \\
  \downarrow \\
  @m \\
  "2"
\end{array} & \begin{array}{c}
  r \\
  \downarrow \\
  C \\
  \downarrow \\
  @m \\
  x \\
  @l \\
  x
\end{array} & \begin{array}{c}
  r \\
  \begin{array}{c}
  A \\
  @l \\
  "1"
  \\
  B \\
  @l \\
  "2"
  \end{array}
\end{array}
\end{array}
\]
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

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

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

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

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

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

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

\[ D \rightarrow E \]
Example: Computing Canonical Solution
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.
  - 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 $CTQ//\text{-query}$.

- Complexity: polynomial time.