XML Data Exchange

Relational Data Exchange Settings

Data Exchange Setting: (σ, τ, Σ)

σ: Source schema.

τ: Target schema.

Σ: Set of rules that specify relationship between the target and the source (source-to-target dependencies).

- Source-to-target dependency:

$$\psi_\tau(\bar{x}, \bar{z}) \coloneq \varphi_\sigma(\bar{x}, \bar{y}).$$

- $\varphi_\sigma(\bar{x}, \bar{y})$: conjunction of atomic formulas over σ.
- $\psi_\tau(\bar{x}, \bar{z})$: conjunction of atomic formulas over τ.

Example: Relational Data Exchange Setting

- $\sigma \quad = \quad Book(Title, AName, Aff)$

- $\tau \quad = \quad Writer(Name, BTitle, Year)$

- $\Sigma \quad = \quad Writer(x_2, x_1, z_1) \text{ :- } Book(x_1, x_2, y_1).$

Relational Data Exchange Problem

- Given a source instance S, find a target instance T such that (S, T) satisfies Σ .
  - (S, T) satisfies $\psi_\tau(\bar{x}, \bar{z}) :- \varphi_\sigma(\bar{x}, \bar{y})$ 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:

S:

Book	*Title*	*AName*	*Aff*
	Algebra	Hungerford	U. Washington
	Real Analysis	Royden	Stanford

Relational Data Exchange Problem

Possible solutions:

T_1:

Writer	*Name*	*BTitle*	*Year*
	Hungerford	Algebra	1974
	Royden	Real Analysis	1988

T_2:

Writer	*Name*	*BTitle*	*Year*
	Hungerford	Algebra	$\perp_1$
	Royden	Real Analysis	$\perp_2$

Query Answering

- Q is a query over target schema.

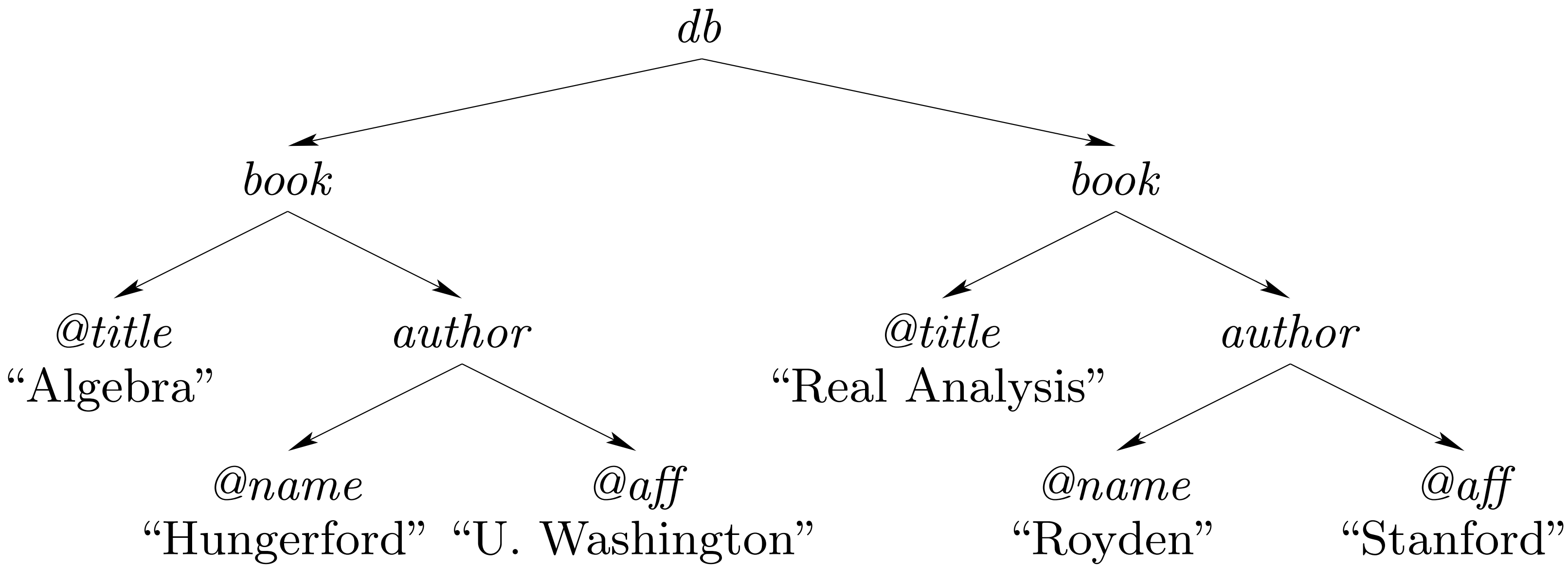
  What does it mean to answer Q?

$$\underline{certain}(Q, S) = \bigcap_{T \text{ is a solution for } S} \; \bigcap_{R \in \mathsf{POSS}(T)} Q(R)$$

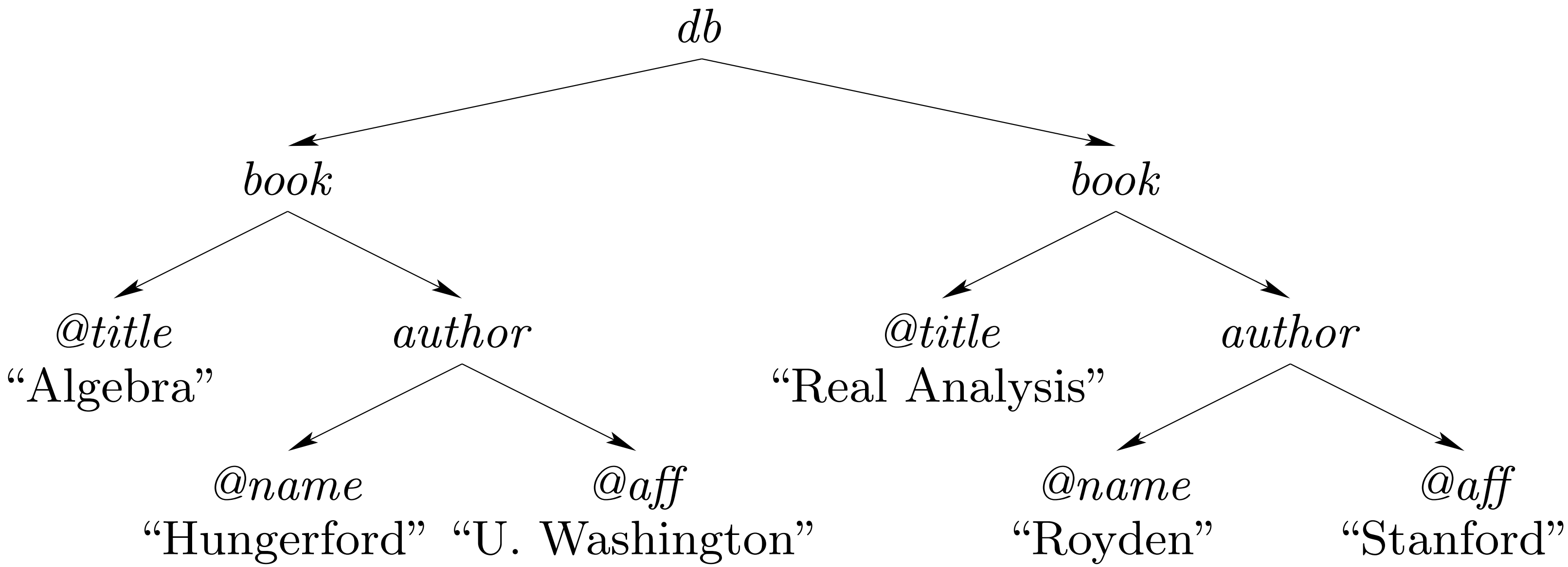
- Previous example:

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

XML Documents



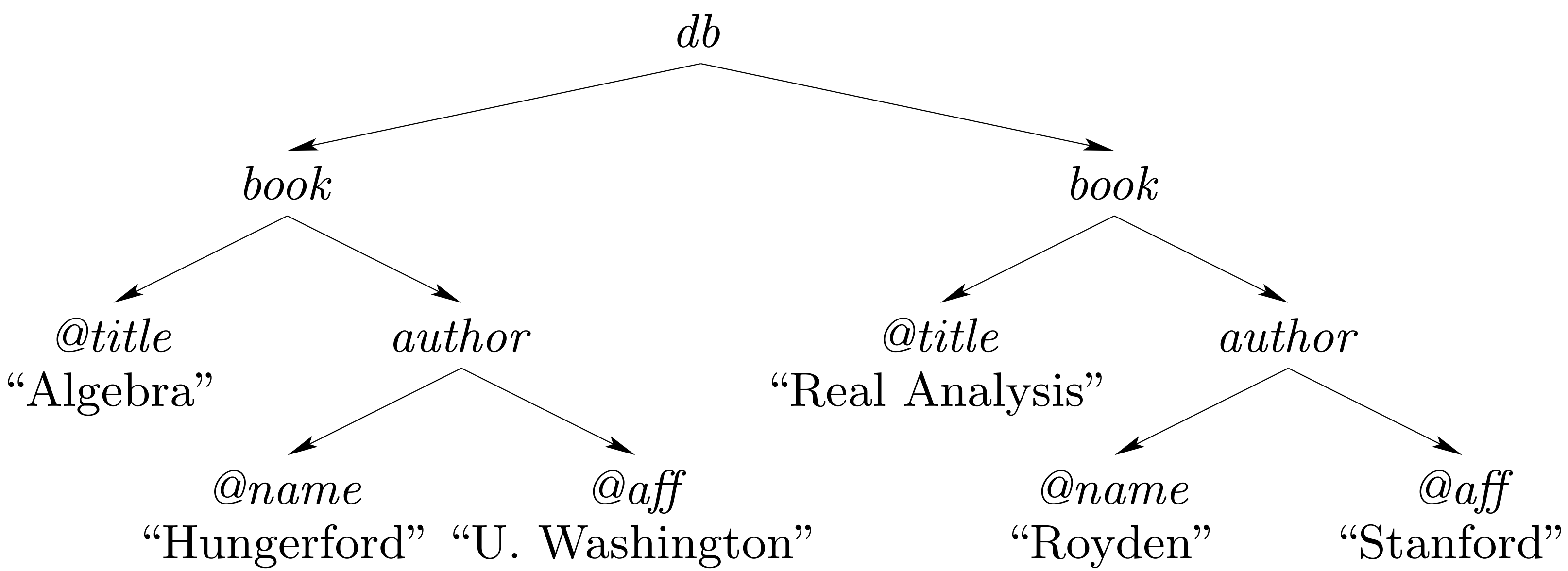
XML Documents



DTD :

$$\begin{array}{rcl} db & \longrightarrow & book^{+} \\ book & \longrightarrow & author^{+} \\ author & \longrightarrow & \varepsilon \end{array}$$

XML Documents



DTD :

$$\begin{array}{rcl} db & \rightarrow & book^{+} \\ book & \rightarrow & author^{+} \\ author & \rightarrow & \varepsilon \end{array} \qquad \begin{array}{rcl} book & \rightarrow & @title \\ author & \rightarrow & @name,\ @aff \end{array}$$

XML Data Exchange Settings

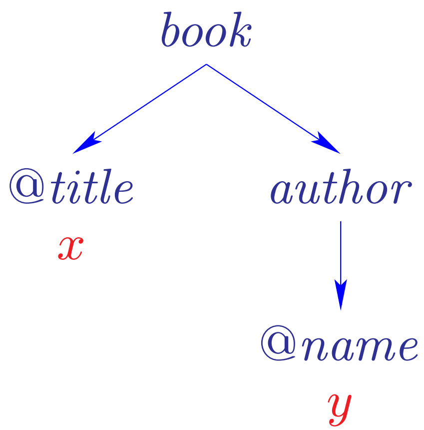
- Instead of source and target relational schemas, we have source and target DTDs.

- But what are the source-to-target dependencies?

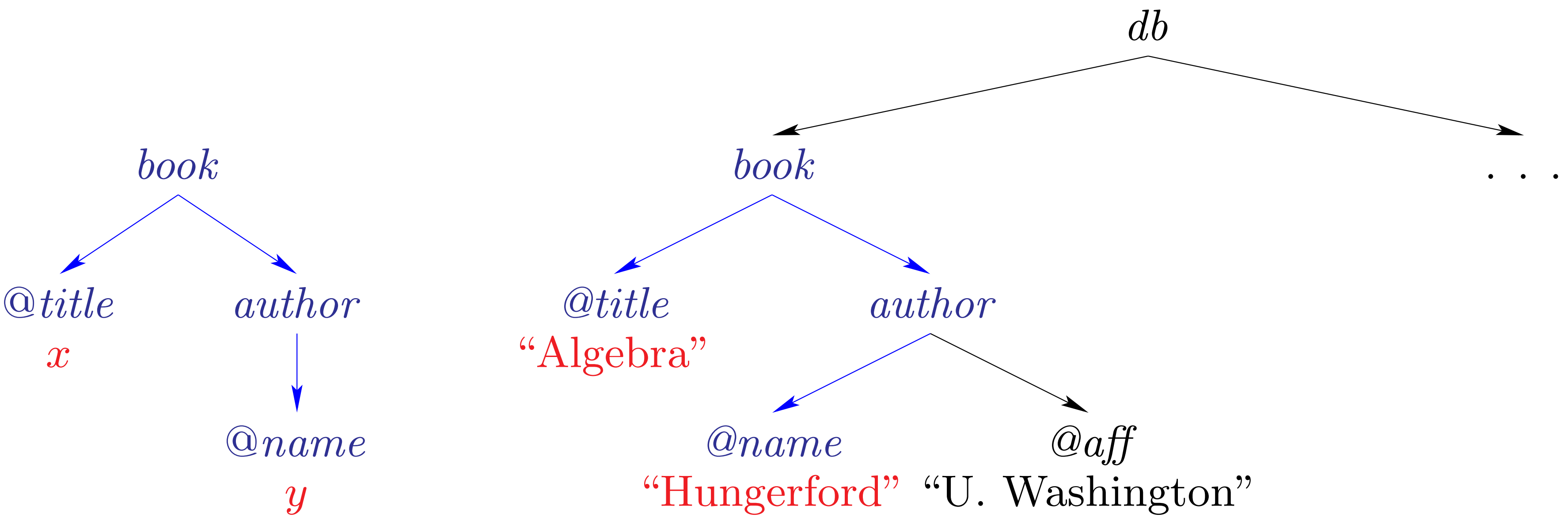
  To define them, we use tree patterns.

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

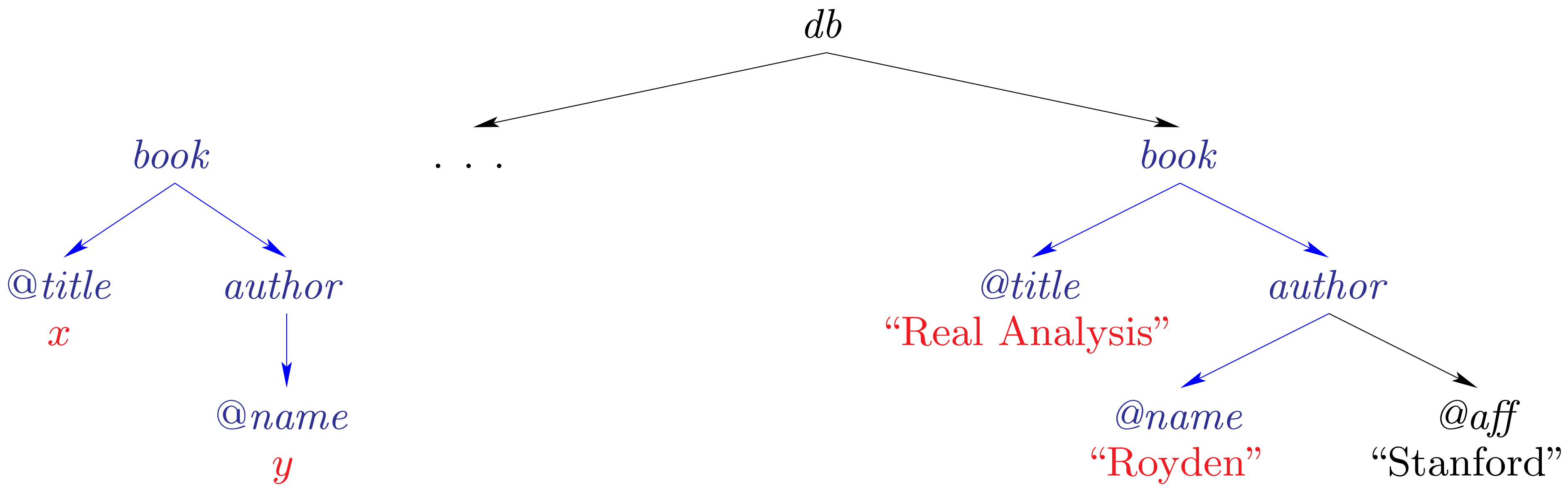
Tree Patterns: Example



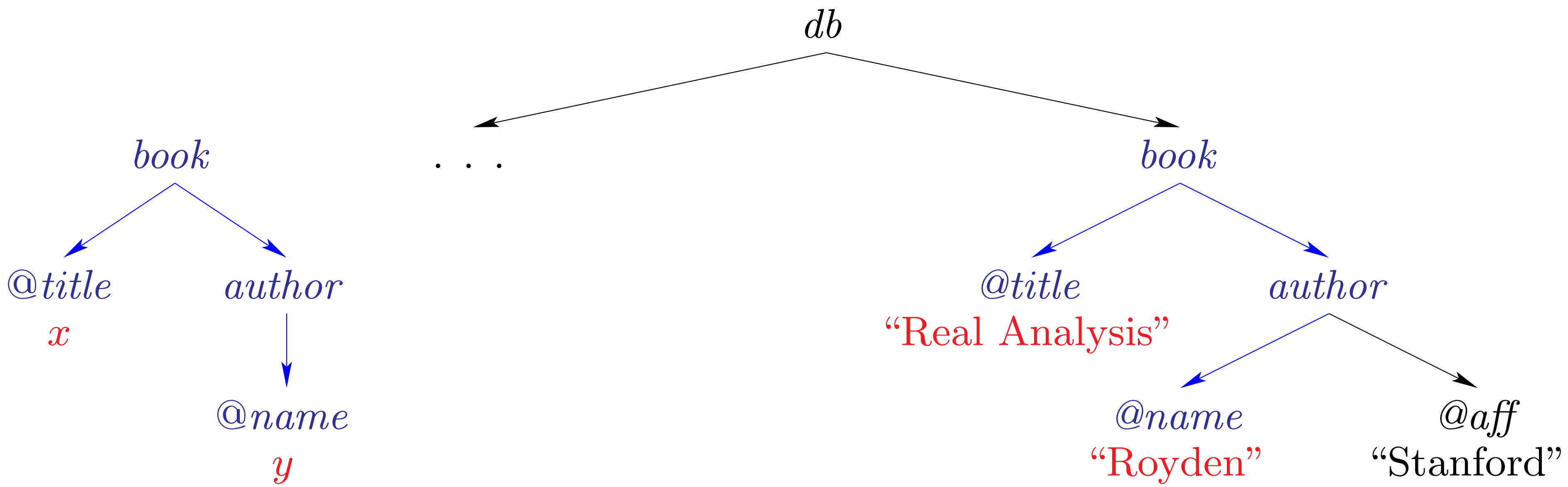
Tree Patterns: Example



Tree Patterns: Example



Tree Patterns: Example



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 $//$.

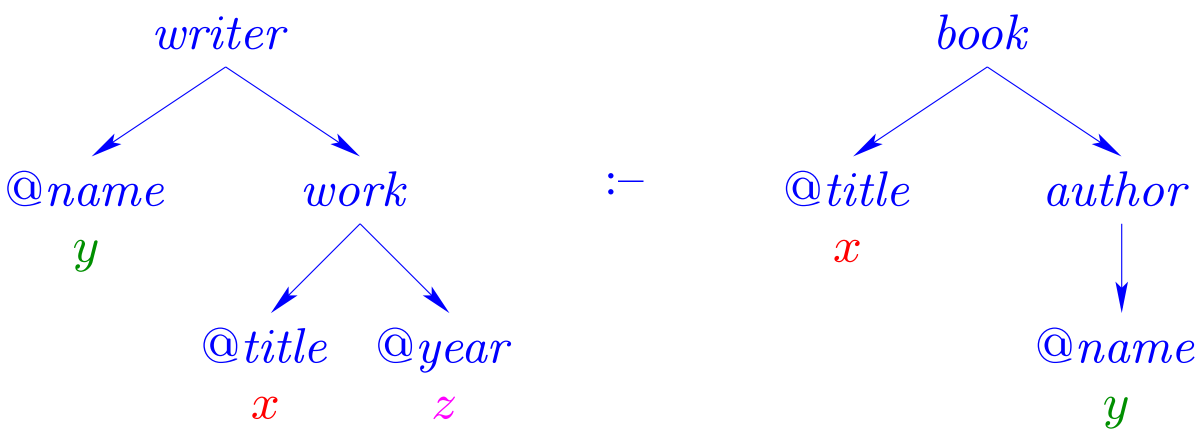
XML Source-to-target Dependencies

- Source-to-target dependency (STD):

$$\psi_\tau(\bar{x}, \bar{z}) \coloneq\!- \ \varphi_\sigma(\bar{x}, \bar{y}),$$

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:



XML Data Exchange Settings

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

D_σ: Source DTD.

D_τ: Target DTD.

Σ : Set of XML source-to-target dependencies.

Each constraint in Σ 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_σ.
- $\psi_\tau(\bar{x}, \bar{z})$: tree-pattern over D_τ.

Example: XML Data Exchange Setting

- Source DTD:

$$\begin{array}{rclcrcl} db & \rightarrow & book^+ & & & & \\ book & \rightarrow & author^+ & \quad & book & \rightarrow & @title \\ author & \rightarrow & \varepsilon & & author & \rightarrow & @name,\ @aff \end{array}$$

- Target DTD:

$$\begin{array}{rclcrcl} bib & \rightarrow & writer^+ & & & & \\ writer & \rightarrow & work^+ & \quad & writer & \rightarrow & @name \\ work & \rightarrow & \varepsilon & & work & \rightarrow & @title,\ @year \end{array}$$

- Σ :

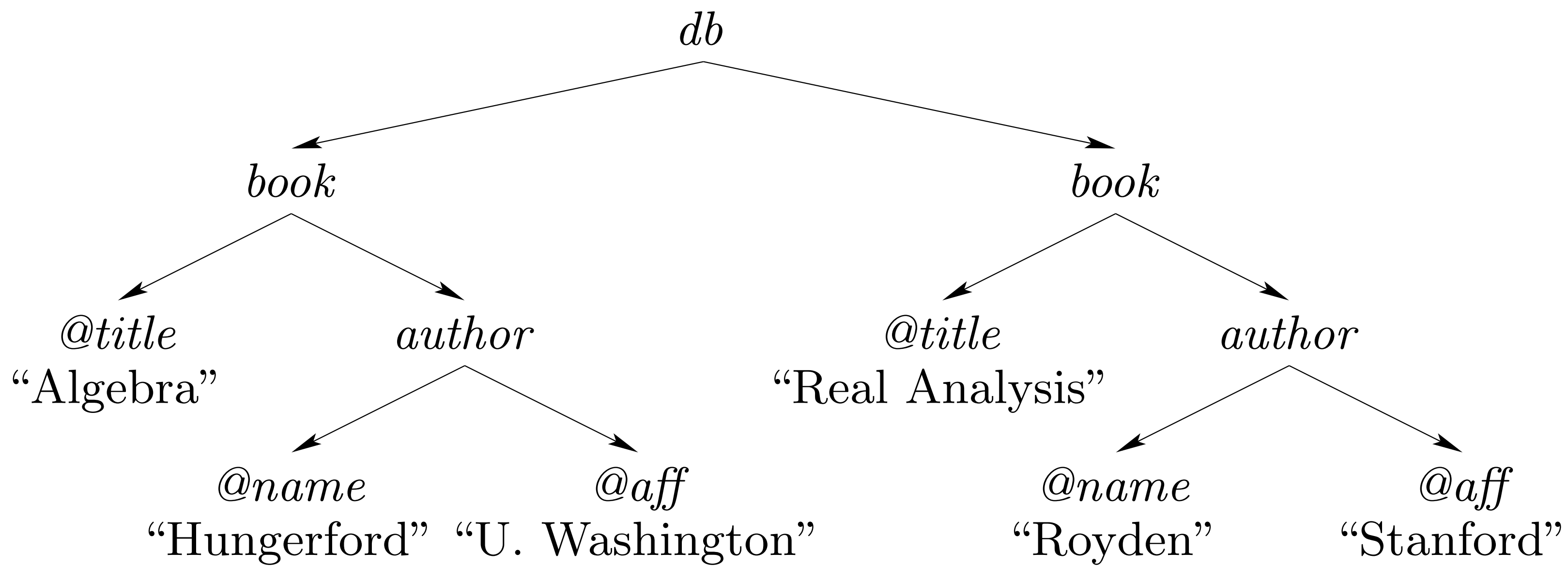
$$writer(@name = y)[work(@title = x, @year = z)] \;\; :- \\ book(@title = x)[author(@name = y)].$$

XML Data Exchange Problem

- Given a source tree T, find a target tree T' such that (T, T') satisfies Σ .
  - (T, T') satisfies $\psi_\tau(\bar{x}, \bar{z})$:– $\varphi_\sigma(\bar{x}, \bar{y})$ 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.

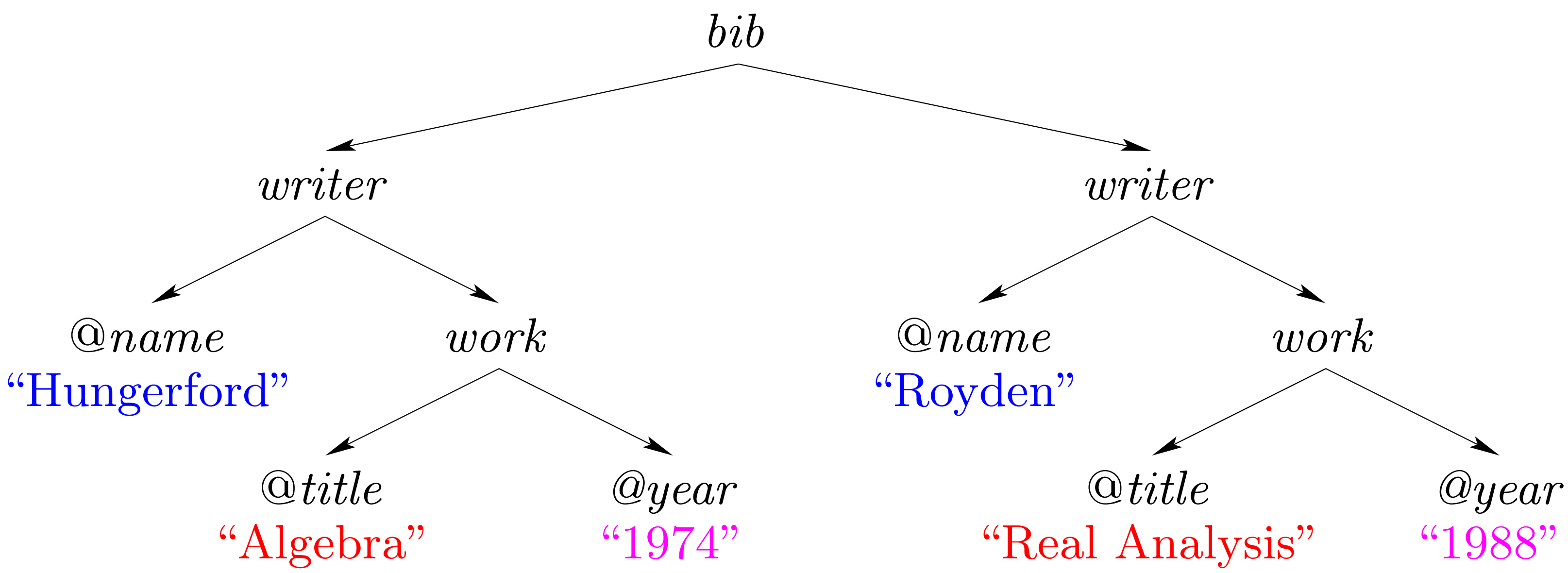
XML Data Exchange Problem

Let T be our original tree:



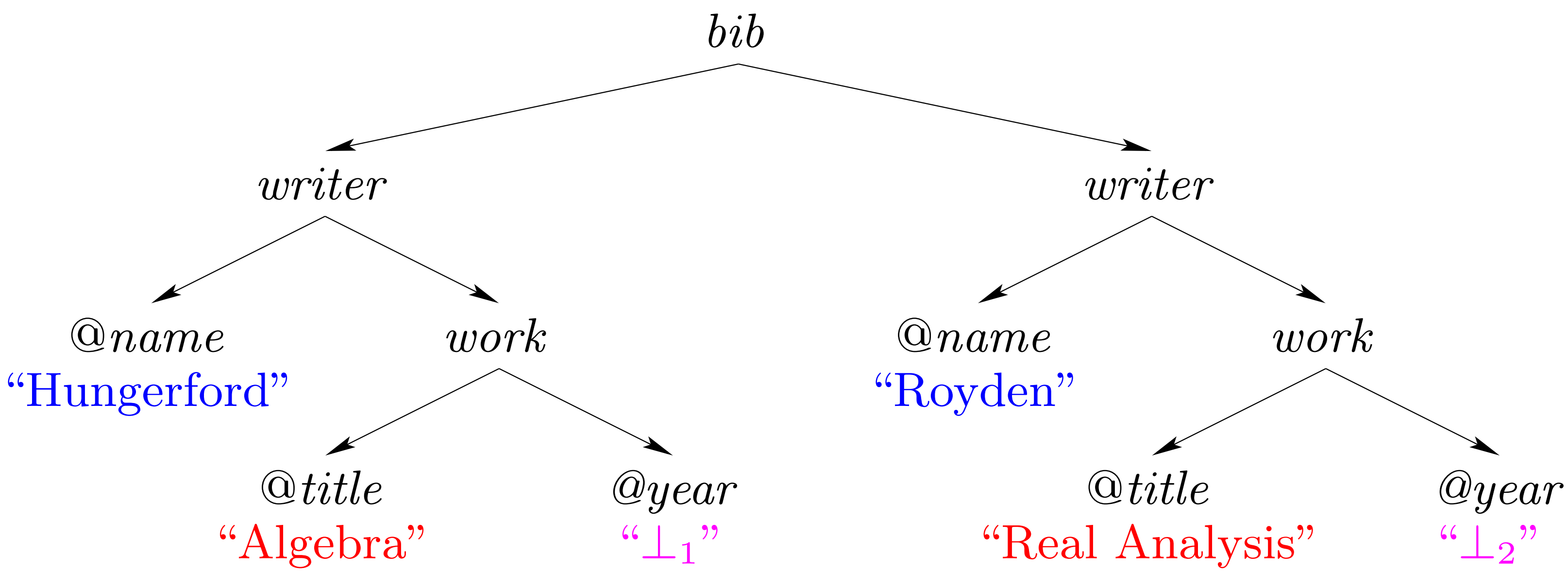
XML Data Exchange Problem

A solution for T:



XML Data Exchange Problem

Another solution for T:



Consistency of XML Data Exchange Settings

- What if we have target DTD

$$\begin{array}{rcl} bib & \to & writer^+ \\ writer & \to & novelist^*, poet^* \\ novelist & \to & work^+ \\ poet & \to & work^+ \\ work & \to & \varepsilon \end{array} \qquad \begin{array}{rcl} & & \\ writer & \to & @name \\ & & \\ & & \\ work & \to & @title, @year \end{array}$$

  in our previous example?

- The setting becomes inconsistent!
  - There are no T conforming to D_σ and T' conforming to D_τ such that (T, T') satisfies Σ .

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.

Complexity-theoretic statement: 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:

$$\ell \;\rightarrow\; \hat{\ell}_1, \ldots, \hat{\ell}_m,$$

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

For example, book $\rightarrow$ title, author$^+$, chapter*, publisher?

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

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.

- For now we use a query language that produces tuples of values.

Conjunctive Tree Queries

- Query language $\mathcal{CTQ}^{//}$ is defined by

$$Q \quad := \quad \varphi \quad | \quad Q \wedge Q \quad | \quad \exists x\, Q,$$

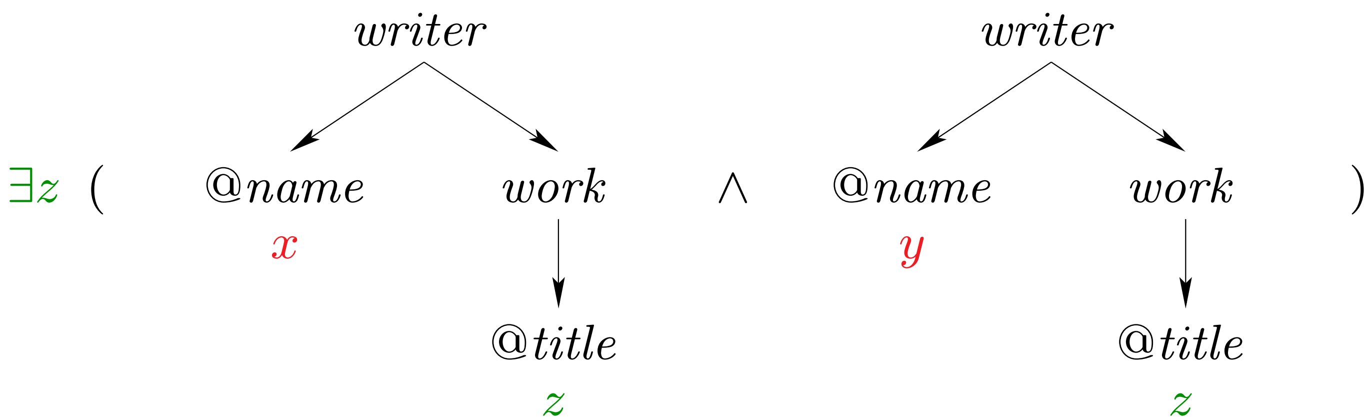
where φ ranges over tree-patterns.

- By disallowing descendant $//$ we obtain restriction $\mathcal{CTQ}$.

Example: Conjunctive Tree Query

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

$$Q(x, y) \;:=$$



Computing Certain Answers

- Semantics: as in the relational case.

$$\underline{certain}(Q,T) \quad = \bigcap_{T' \text{ is a solution for } T} Q(T').$$

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

PROBLEM:	$\textsc{CertAnsw}(Q)$.
INPUT:	Tree T conforming to D_σ and tuple $\bar{a}$.
QUESTION:	Is $\bar{a} \in \underline{certain}(Q,T)$?

Computing Certain Answers: General Picture

It is not even clear if the problem is solvable.

Good news For every XML data exchange setting and $\mathcal{CTQ}^{//}$-query Q, th problem $\textsc{CertAnsw}(Q)$ is solvable in exponential time.

More precisely, it is in coNP.

Not so good news Sometimes exponential time is "unavoidable": There exist an XML data exchange setting and a $\mathcal{CTQ}^{//}$-query Q such that $\textsc{CertAnsw}(Q)$ is 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 $\textsc{CertAnsw}(Q)$ 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 $\underline{certain}(Q, S) = \text{certain}(Q, \textsc{CanSol}(S))$.
- Idea: given a source tree T, compute a solution $T^\star$ for T such that

$$\underline{certain}(Q, T) \;=\; remove\_null\_tuples(Q(T^\star)).$$

- $T^\star$ is a canonical solution for T.
- We compute $T^\star$ in two steps:
  - We use STDs to compute a canonical pre-solution $cps(T)$ from T.
  - Then we use target DTD to compute $T^\star$ from $cps(T)$.

Example: XML Data Exchange Setting

- Source DTD:

$$\begin{array}{lllcllll} r & \rightarrow & A^*, B^* & & & & \\ A & \rightarrow & \varepsilon & & A & \rightarrow & @\ell \\ B & \rightarrow & \varepsilon & & B & \rightarrow & @\ell \end{array}$$

- Target DTD:

$$\begin{array}{lllcllll} r & \rightarrow & (C, D)^* & & & & \\ C & \rightarrow & \varepsilon & & C & \rightarrow & @m \\ D & \rightarrow & E & & & & \\ E & \rightarrow & \varepsilon & & E & \rightarrow & @n \end{array}$$

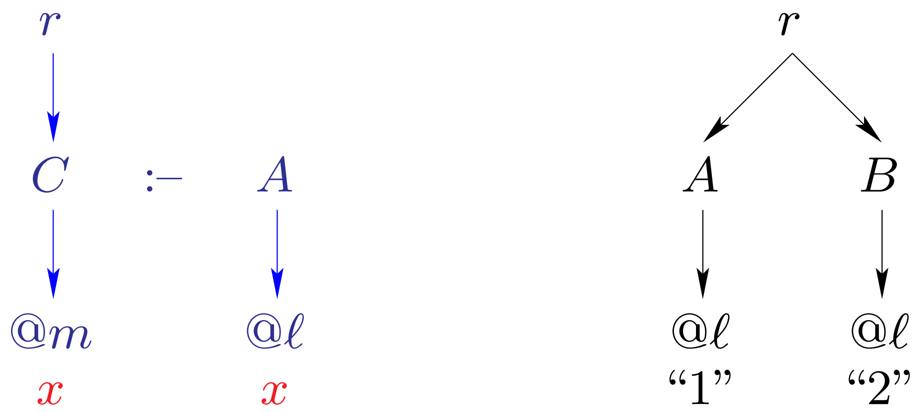
- Σ :

$$\begin{array}{lcl} r[C(@m = x)] & :- & A(@\ell = x), \\ r[C(@m = x)] & :- & B(@\ell = x). \end{array}$$

Example: Computing Canonical Pre-solution

$$
\begin{tikzcd}
 & r \arrow[dl] \arrow[dr] & \\
A \arrow[d] & & B \arrow[d] \\
\begin{array}{c} @\ell \\ \text{``1''} \end{array} & & \begin{array}{c} @\ell \\ \text{``2''} \end{array}
\end{tikzcd}
$$

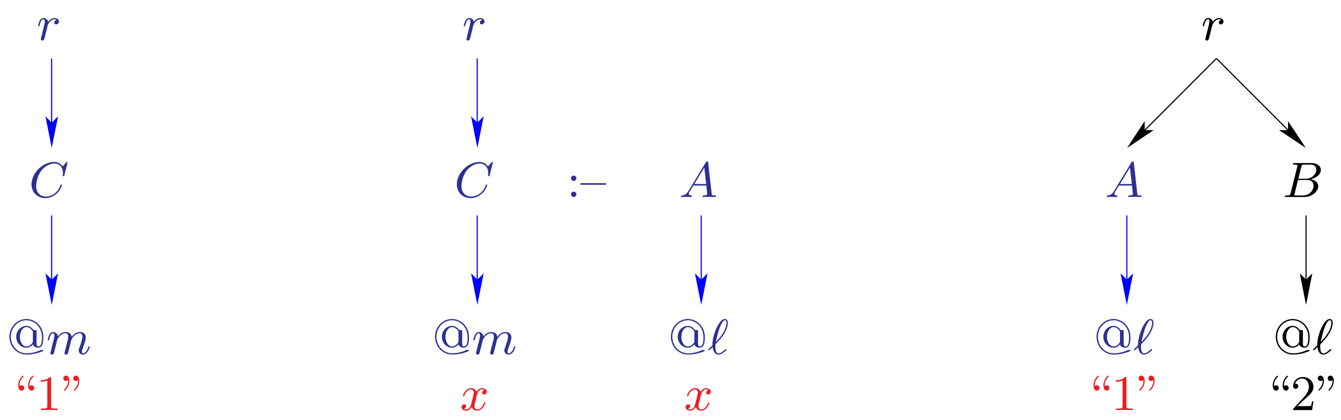
Example: Computing Canonical Pre-solution



Example: Computing Canonical Pre-solution

$$
\begin{tikzcd}
r \arrow[d] & & \\
C \arrow[d] & {:-} & A \arrow[d] \\
@m & & @\ell \\
x & & x
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
& r \arrow[dl] \arrow[dr] & \\
A \arrow[d] & & B \arrow[d] \\
@\ell & & @\ell \\
\text{“1”} & & \text{“2”}
\end{tikzcd}
$$

Example: Computing Canonical Pre-solution



Example: Computing Canonical Pre-solution

$$\begin{array}{c} r \\ \downarrow \\ C \\ \downarrow \\ @m \\ \text{“1”} \end{array}$$

Example: Computing Canonical Pre-solution

$$
\begin{tikzcd}
r \arrow[d] \\
C \arrow[d] \\
\begin{array}{c} @m \\ \text{“1”} \end{array}
\end{tikzcd}
$$

$$
\begin{tikzcd}
 & r \arrow[dl] \arrow[dr] & \\
A \arrow[d] & & B \arrow[d] \\
\begin{array}{c} @\ell \\ \text{“1”} \end{array} & & \begin{array}{c} @\ell \\ \text{“2”} \end{array}
\end{tikzcd}
$$

Example: Computing Canonical Pre-solution

$$
\begin{tikzcd}[row sep=small]
r \arrow[d] \\
C \arrow[d] \\
@m \\[-2em]
\text{“1”}
\end{tikzcd}
$$

$$
\begin{tikzcd}[row sep=small]
r \arrow[d] & & \\
C \arrow[d] & :- & B \arrow[d] \\
@m & & @\ell \\[-2em]
x & & x
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}[row sep=small]
& r \arrow[dl] \arrow[dr] & \\
A \arrow[d] & & B \arrow[d] \\
@\ell & & @\ell \\[-2em]
\text{“1”} & & \text{“2”}
\end{tikzcd}
$$

Example: Computing Canonical Pre-solution

$$
\begin{tikzcd}
r \arrow[d] \\
C \arrow[d] \\
@m \\[-2em]
\text{“1”}
\end{tikzcd}
$$

$$
\begin{tikzcd}
r \arrow[d] & & \\
C \arrow[d] & :- & B \arrow[d] \\
@m & & @\ell \\[-2em]
x & & x
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
& r \arrow[dl] \arrow[dr] & \\
A \arrow[d] & & B \arrow[d] \\
@\ell & & @\ell \\[-2em]
\text{“1”} & & \text{“2”}
\end{tikzcd}
$$

Example: Computing Canonical Pre-solution

$$
\begin{tikzcd}
r \arrow[d] \\
C \arrow[d] \\
@m \\
\text{“1”}
\end{tikzcd}
$$

$$
\begin{tikzcd}
r \arrow[d] \\
C \arrow[d] \\
@m \\
\text{“2”}
\end{tikzcd}
\qquad
\begin{tikzcd}
r \arrow[d] & & \\
C \arrow[d] & :- & B \arrow[d] \\
@m & & @\ell \\
x & & x
\end{tikzcd}
\qquad
\begin{tikzcd}
& r \arrow[dl] \arrow[dr] & \\
A \arrow[d] & & B \arrow[d] \\
@\ell & & @\ell \\
\text{“1”} & & \text{“2”}
\end{tikzcd}
$$

Example: Computing Canonical Pre-solution

$$\begin{tikzcd} r \arrow[d] \\ C \arrow[d] \\ @m \end{tikzcd}$$

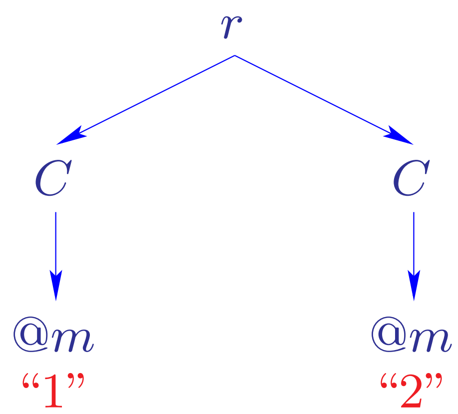
"1"

$$\begin{tikzcd} r \arrow[d] \\ C \arrow[d] \\ @m \end{tikzcd}$$

"2"

Example: Computing Canonical Pre-solution

Canonical pre-solution:

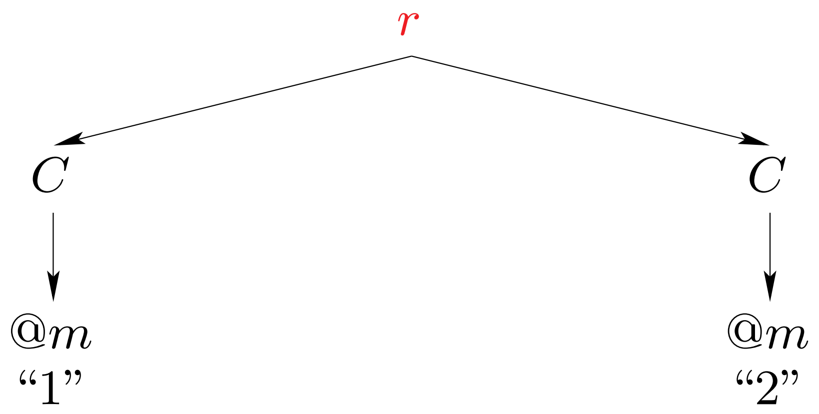


Not yet a solution: it does not conform to the target DTD.

Example: Computing Canonical Solution

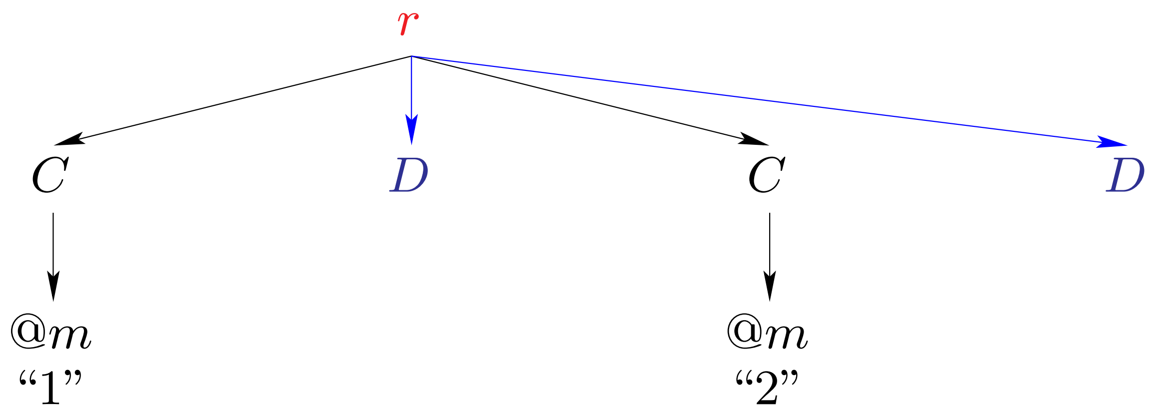
$$
\begin{array}{ccc}
 & r & \\
\swarrow & & \searrow \\
C & & C \\
\downarrow & & \downarrow \\
@m & & @m \\
\text{“1”} & & \text{“2”}
\end{array}
$$

Example: Computing Canonical Solution



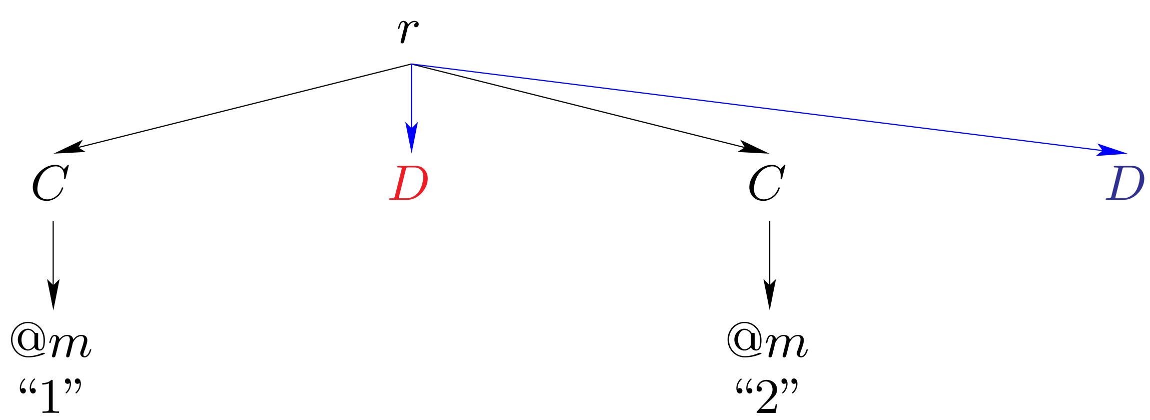
$$r \quad \longrightarrow \quad (C, D)^*$$

Example: Computing Canonical Solution



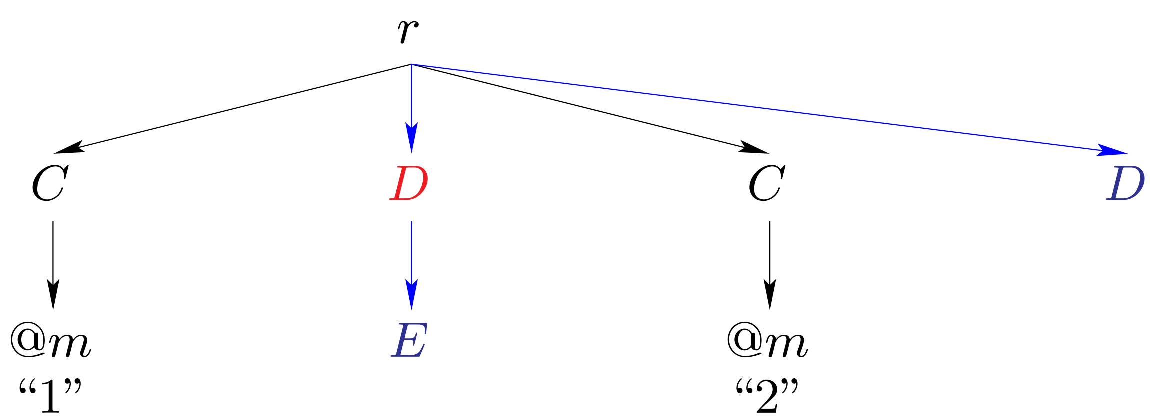
$$r \quad \longrightarrow \quad (C, D)^*$$

Example: Computing Canonical Solution



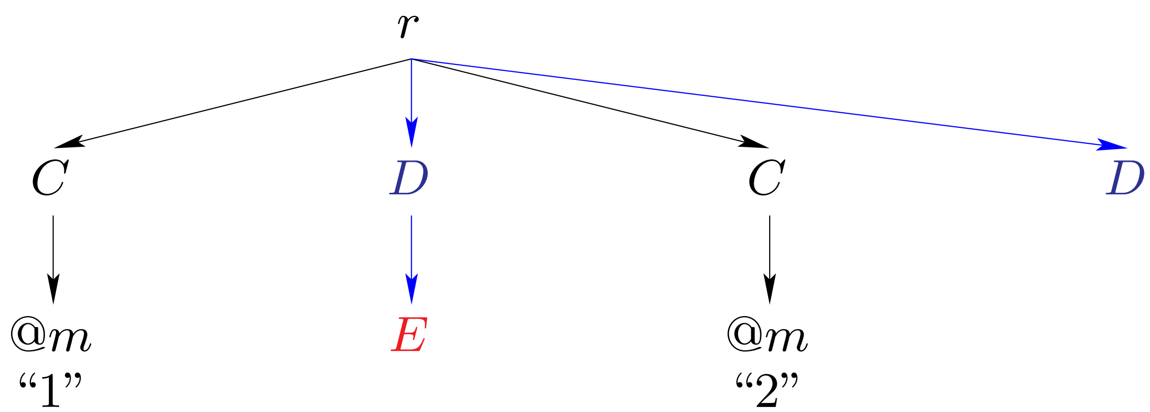
$$D \longrightarrow E$$

Example: Computing Canonical Solution



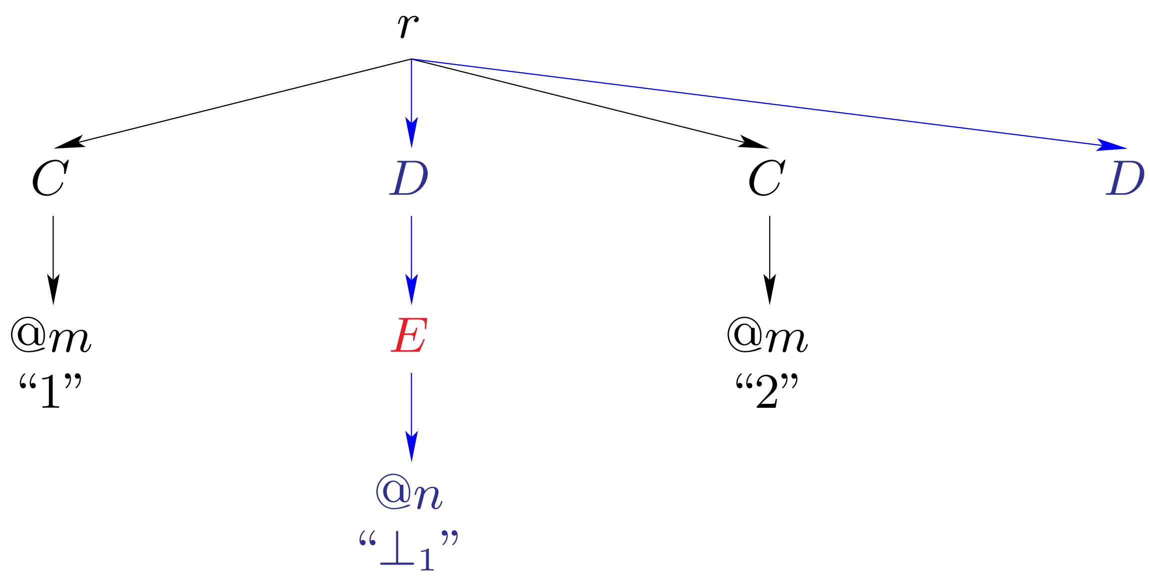
$$D \longrightarrow E$$

Example: Computing Canonical Solution



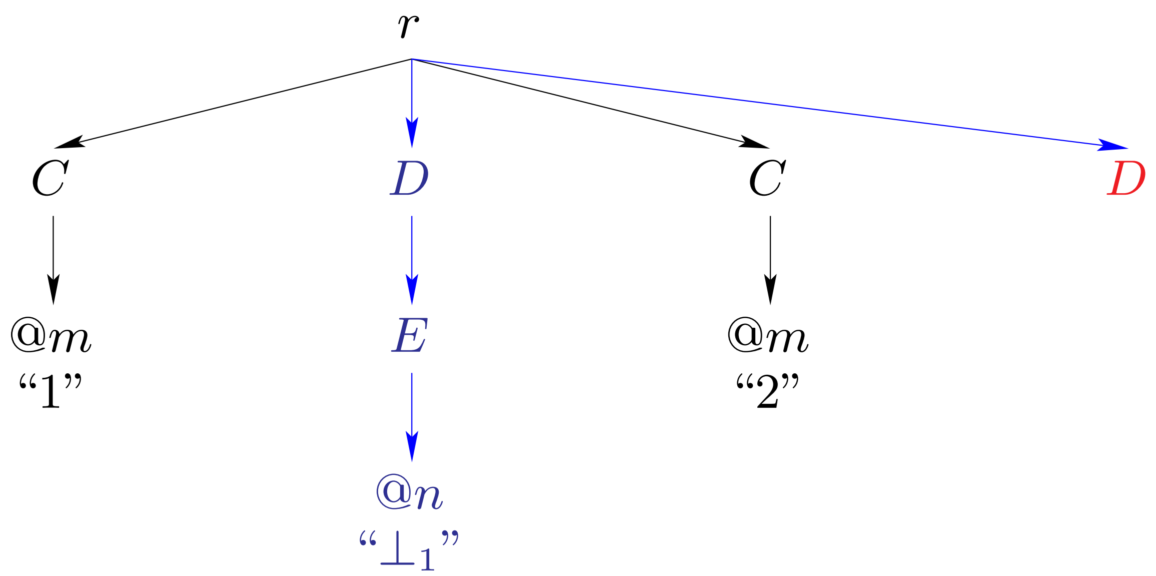
$$E \longrightarrow @n$$

Example: Computing Canonical Solution



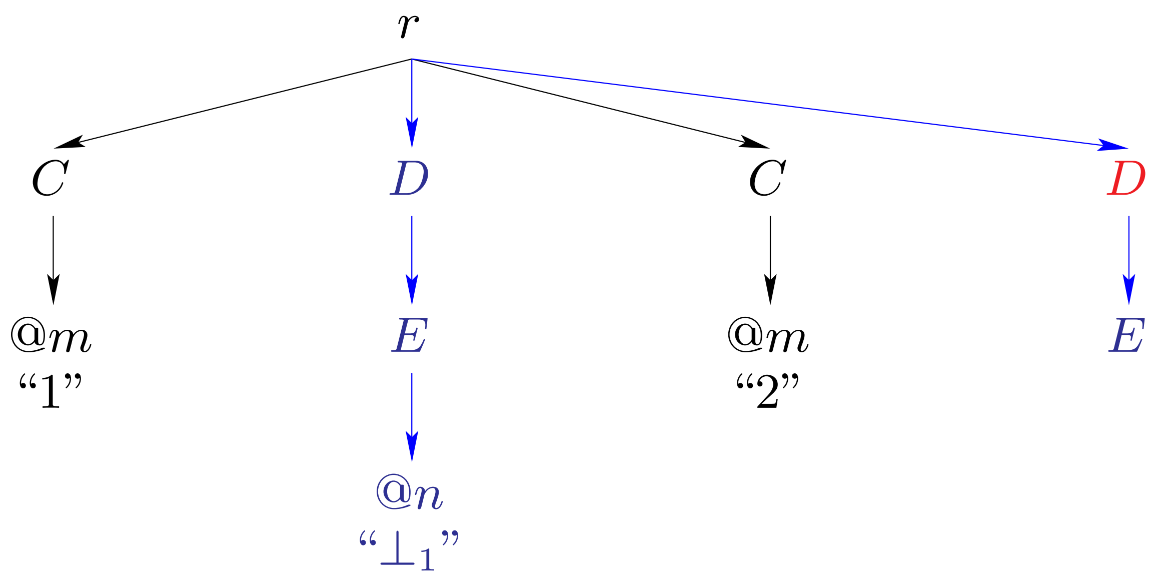
$$E \longrightarrow @n$$

Example: Computing Canonical Solution



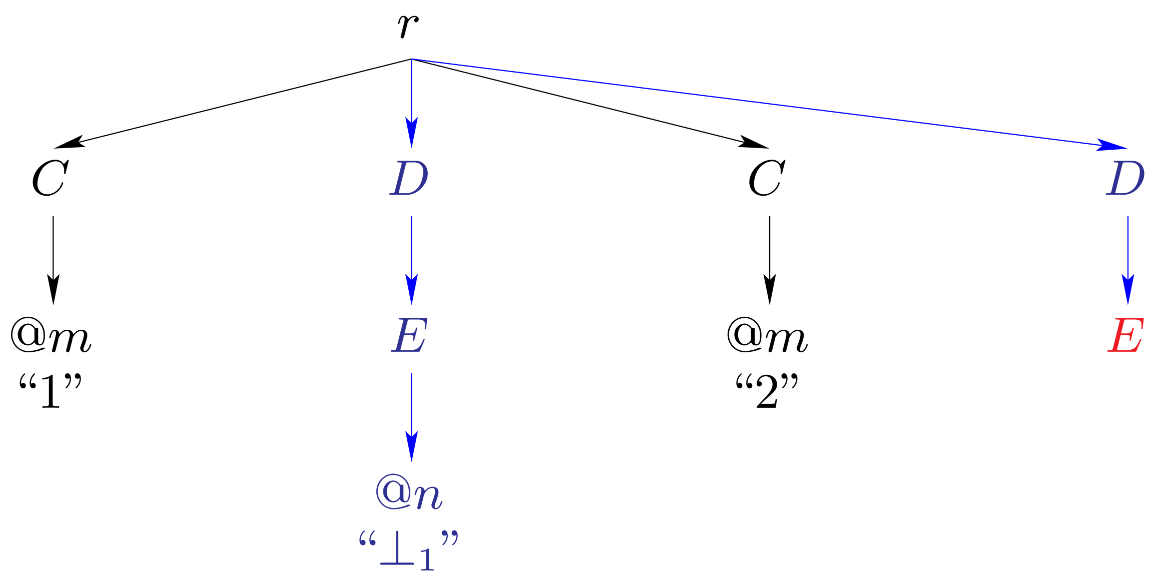
$$D \quad \rightarrow \quad E$$

Example: Computing Canonical Solution



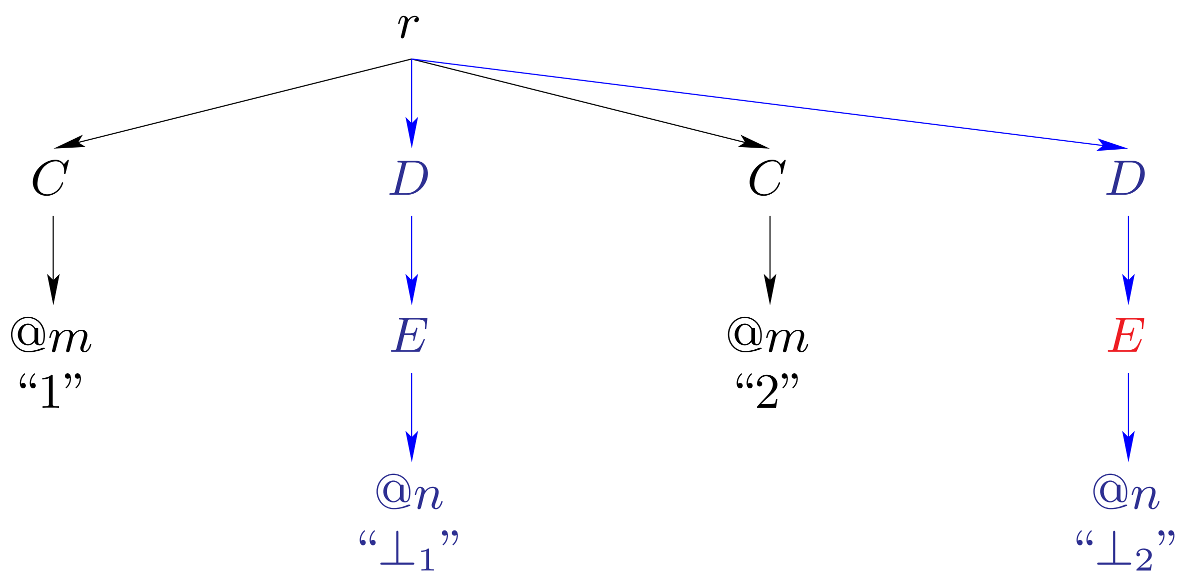
$$D \longrightarrow E$$

Example: Computing Canonical Solution



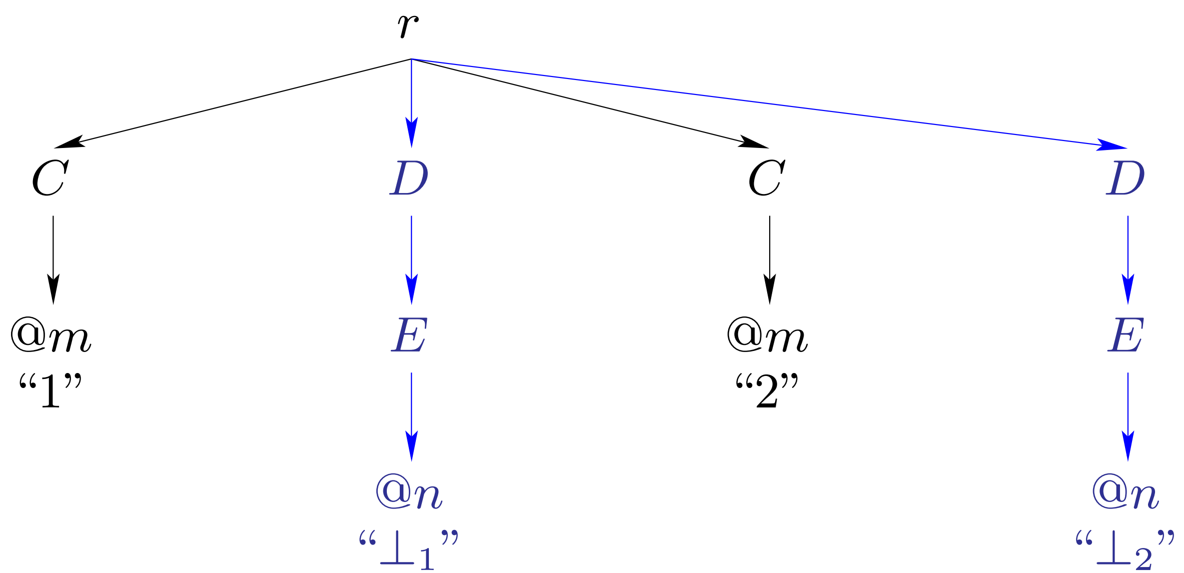
$$E \quad \longrightarrow \quad @n$$

Example: Computing Canonical Solution



$$E \quad \longrightarrow \quad @n$$

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 $T^\star$ for T.
  - Previous algorithm computes canonical solution $T^\star$ for T in polynomial time.
  - $\underline{certain}(Q, T) = remove\_null\_tuples(Q(T^\star))$, for every $\mathcal{CTQ}^{//}$-query.

- Complexity: polynomial time.