Theory of Peer Data Management

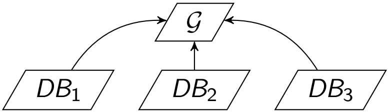
Sebastian Skritek

Database and Artificial Intelligence Group
Vienna University of Technology

DEIS 2010

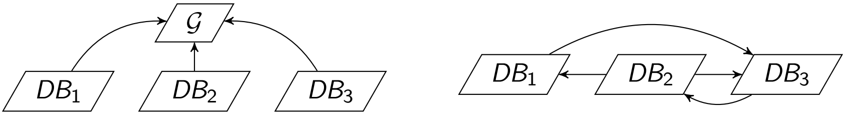
Motivation

From Data Integration to Peer Data Integration



Motivation

From Data Integration to Peer Data Integration



Extend semantics from data integration, BUT:

- query answering may become undecidable
- some tractable fragments are very restrictive
- further undesired properties

$\Rightarrow$ several suggestions made for semantics of mappings

Motivation

$\Rightarrow$ use "tools" from data exchange and data integration
(but they are not completely satisfactory)

Additional problems (compared to DEI)

- Modularity of peers
- Inconsistencies, Updates
- Trust

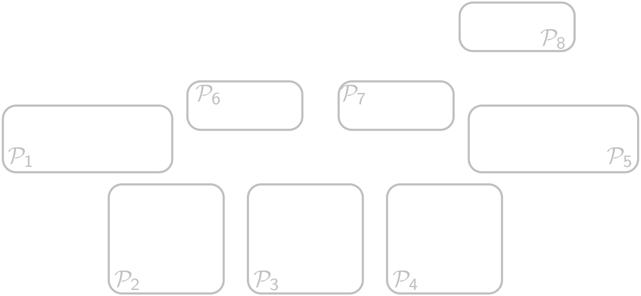
Peer Data Management covers a variety of scenarios
$\Rightarrow$ take a look on the theory behind some of these systems
(formal semantics, decidability, complexity, ...)

Talk Outline

1. Motivation
2. Query Answering in Peer Data Management
3. Materialization of Data in Peer Data Management
4. Optimization of Query Reformulation
5. Conclusion

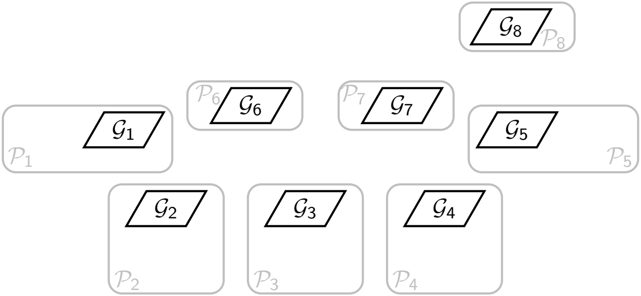
Outline

A Framework for Peer Data Integration



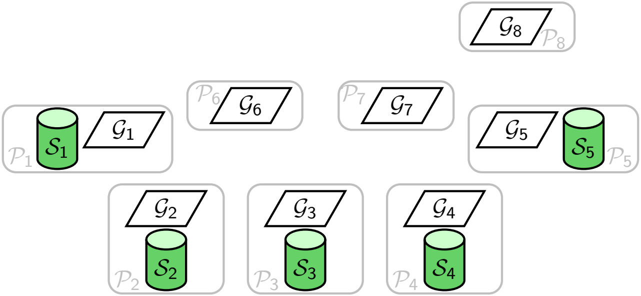
- Each peer $\mathcal{P} = (\mathcal{G}, \mathcal{S}, \mathcal{L}, \mathcal{M})$ consists of

A Framework for Peer Data Integration



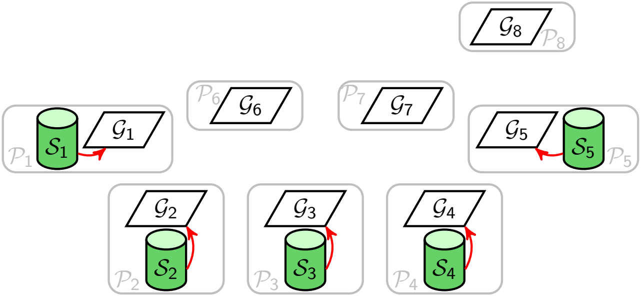
- Each peer $\mathcal{P} = (\mathcal{G}, \mathcal{S}, \mathcal{L}, \mathcal{M})$ consists of
  - a peer schema $\mathcal{G}$

A Framework for Peer Data Integration



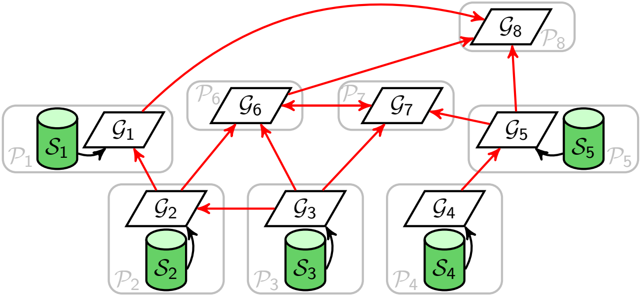
- Each peer $\mathcal{P} = (\mathcal{G}, \mathcal{S}, \mathcal{L}, \mathcal{M})$ consists of
  - a peer schema $\mathcal{G}$
  - a (possible empty) local/source schema $\mathcal{S}$

A Framework for Peer Data Integration



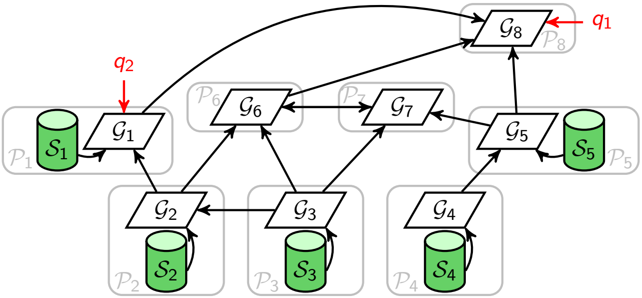
- Each peer $\mathcal{P} = (\mathcal{G}, \mathcal{S}, \mathcal{L}, \mathcal{M})$ consists of
  - a peer schema $\mathcal{G}$
  - a (possible empty) local/source schema $\mathcal{S}$
  - a (possible empty) set of local mappings $\mathcal{L}$: $\{cq_{\mathcal{S}} \rightsquigarrow cq_{\mathcal{G}}\}$

A Framework for Peer Data Integration



- Each peer $\mathcal{P} = (\mathcal{G}, \mathcal{S}, \mathcal{L}, \mathcal{M})$ consists of
  - a peer schema $\mathcal{G}$
  - a (possible empty) local/source schema $\mathcal{S}$
  - a (possible empty) set of local mappings $\mathcal{L}$: $\{cq_{\mathcal{S}} \rightsquigarrow cq_{\mathcal{G}}\}$
  - a set of peer mappings $\mathcal{M}$: $\{cq_{\mathcal{P}'} \rightsquigarrow cq_{\mathcal{P}}\}$

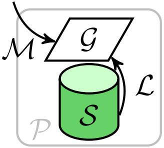
A Framework for Peer Data Integration



- Each peer $\mathcal{P} = (\mathcal{G}, \mathcal{S}, \mathcal{L}, \mathcal{M})$ consists of
  - a peer schema $\mathcal{G}$
  - a (possible empty) local/source schema $\mathcal{S}$
  - a (possible empty) set of local mappings $\mathcal{L}$: $\{cq_{\mathcal{S}} \rightsquigarrow cq_{\mathcal{G}}\}$
  - a set of peer mappings $\mathcal{M}$: $\{cq_{\mathcal{P}'} \rightsquigarrow cq_{\mathcal{P}}\}$
- Queries q are posed over peer schema of a single peer
  - data remains in sources, queries (and results) are propagated

$\mathcal{PPL}$ (Peer Programming Language)

[Halevy et al., VLDB J. 2005]



$\mathcal{P} = (\mathcal{G}, \mathcal{S}, \mathcal{L}, \mathcal{M})$

Definition

- Local Mappings $\mathcal{L}$:
  - $\mathcal{P}\colon r \subseteq cq$ $(\mathcal{P}\colon r = cq)$
- Peer Mappings $\mathcal{M}$:
  - $cq'_{\mathcal{P}'} \subseteq cq_{\mathcal{P}}$ $(cq'_{\mathcal{P}'} = cq_{\mathcal{P}})$
    inclusion/equality mappings
  - $r_{\mathcal{P}}(\vec{x})$:- $cq_{\mathcal{P}'}(\vec{x})$
    definitional mappings
- $\mathcal{G}$, $\mathcal{S}$:
  - relational schemas

(Note: mappings only between pairs of peers)

$\mathcal{PPL}$: Semantics

Definition (consistent data instance)

Let N be a PDMS, D an instance for $\mathcal{S}$.
Instance I for $\mathcal{G}$ is consistent with N and D if

- for every $m \in \mathcal{L}$
  - $r^D \subseteq cq^I$ (resp. $r^D = cq^I$)
- for every $m \in \mathcal{M}$ either
  - $cq'^I_{\mathcal{P}'} \subseteq cq^I_{\mathcal{P}}$ (resp. $cq'^I_{\mathcal{P}'} = cq^I_{\mathcal{P}}$) or
  - $r(\vec{x})^I = body(m_1)^I \cup \cdots \cup body(m_n)^I$ where $r = head(m)$, and $\{m \in \mathcal{M} \mid head(m) = r\} = \{m_1, \ldots, m_n\}$

certain answers to query $q(\vec{x})$: tuples $\vec{t}$ s.t. $\vec{t} \in q(\vec{x})^I$ for every consistent instance I

$\mathcal{PPL}$: First Order Interpretation

$$\text{PDMS } \mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$$

(consider only inclusion storage descriptions)

→ Define semantics in terms of FO logic:

- $\forall \vec{x}(r(\vec{x}) \rightarrow \exists \vec{z} \psi_{\mathcal{G}}(\vec{x}, \vec{z}))$ (for each $m \in \mathcal{L}_i$)
- $\forall \vec{x}(\exists \vec{y}(\phi_{\mathcal{P}_i}(\vec{x}, \vec{y})) \rightarrow \exists \vec{z} \psi_{\mathcal{P}_j}(\vec{x}, \vec{z}))$ (for each $m \in \mathcal{M}$)
  - allow only restricted inclusion peer mappings,
  - use disjunctive TGDs for definitional mappings

⇒ certain answers w.r.t. D: answer in every model I of $\mathcal{P}$

- FO theory $T_{\mathcal{P}_i}$ for $\mathcal{P}_i$, $T_{\mathcal{P}} = \bigcup_{\mathcal{P}_i \in \mathcal{P}} T_{\mathcal{P}_i}$ for $\mathcal{P}$
- Models for theories (given instances D_i for $\mathcal{S}_i$):
  - Model of $T_{\mathcal{P}_i}$ ($T_{\mathcal{P}}$) based on D_i ($D = \bigcup_i D_i$):
    - interpretation I of $T_{\mathcal{P}_i}$ ($T_{\mathcal{P}}$) s.t. $s^I = s^{D_i}$ (for each $s \in \mathcal{S}_i$)
  - Model of $\mathcal{P}$ based on D:
    - model of $T_{\mathcal{P}}$ based on D and of mappings $\mathcal{M}$

$\mathcal{PPL}$: Complexity

Theorem (Halevy et al., 2005)

Let N be a PDMS specified in $\mathcal{PPL}$

1. *Finding all certain answers to CQ q is undecidable*
2. *If N contains only inclusion peer and storage descriptions and the peer mappings are acyclic*
   $\Rightarrow$ *CQ answering in polynomial time (data complexity)*

Proof (sketch).

(2) Encode query and mappings in a nonrecursive datalog program with Skolem terms $\Rightarrow$ evaluation PTIME (data complexity). $\square$

$\mathcal{PPL}$: Complexity (contd.)

Finding all certain answers to a CQ q is undecidable

Proof (sketch).

(1) Shown by reduction from implication problem for FDs and IDs:
Given $\vec{R}$, Σ, $\varphi = R_i[A] \subseteq R_j[B]$
$\Rightarrow N = \{\mathcal{P}_1\}$, with $\mathcal{P}_1 = (\vec{R}, \{S/1\}, \{S \subseteq R_i[A]\}, \mathcal{M})$, and $\mathcal{M}$:

- for FD $R_i\colon \vec{A} \to B$:
  $\{(\vec{A}, B_1, B_2) \mid R_i(\vec{A}, B_1), R_i(\vec{A}, B_2)\} \subseteq \{(\vec{A}, B, B) \mid R_i(\vec{A}, B)\}$
- for ID $R_i[\vec{A}] \subseteq R_j[\vec{B}]$: $R_i[\vec{A}] \subseteq R_j[\vec{B}]$

Then let $I = \{S(a)\}$, and $q\colon \{R_j[B]\}$.

It holds that $\Sigma \models \varphi$ iff q returns a. □

$\mathcal{PPL}$: Complexity (contd.)

Consider the following restrictions:

1. equality storage or peer mappings do not contain projection
2. peer relations that appear in the head of a definitional mapping do not appear on the rhs of any other mapping

Theorem (Halevy et al., VLDB J. 2005)

All inclusion peer mappings acyclic, but equality peer mappings $\Rightarrow$ CQ answering is (data complexity)

- *If (1) and (2) $\Rightarrow$ in* PTIME
- *If (1) but not (2) $\Rightarrow$* coNP *complete*
- *If (2) but not (1) $\Rightarrow$* coNP *complete*

$\mathcal{PPL}$: Query Answering

Consider again the mappings:

- Peer Mappings $\mathcal{M}$:
  - $cq'_{\mathcal{P}'} = cq_{\mathcal{P}} \Rightarrow cq'_{\mathcal{P}'} \subseteq cq_{\mathcal{P}}$ and $cq_{\mathcal{P}} \subseteq cq'_{\mathcal{P}'}$
  - $cq'_{\mathcal{P}'} \subseteq cq_{\mathcal{P}} \Rightarrow v(\vec{x}) \subseteq cq_{\mathcal{P}}$ and $v(\vec{x})$:- $cq'_{\mathcal{P}'}$
  - $r_{\mathcal{P}}(\vec{x})$:- $cq_{\mathcal{P}'}(\vec{x})$

$\Rightarrow$ pure LAV and GAV mappings

$\mathcal{PPL}$: Query Answering

Consider again the mappings:

- Peer Mappings $\mathcal{M}$:
  - $cq'_{\mathcal{P}'} = cq_{\mathcal{P}} \Rightarrow cq'_{\mathcal{P}'} \subseteq cq_{\mathcal{P}}$ and $cq_{\mathcal{P}} \subseteq cq'_{\mathcal{P}'}$
  - $cq'_{\mathcal{P}'} \subseteq cq_{\mathcal{P}} \Rightarrow v(\vec{x}) \subseteq cq_{\mathcal{P}}$ and $v(\vec{x})$:- $cq'_{\mathcal{P}'}$
  - $r_{\mathcal{P}}(\vec{x})$:- $cq_{\mathcal{P}'}(\vec{x})$

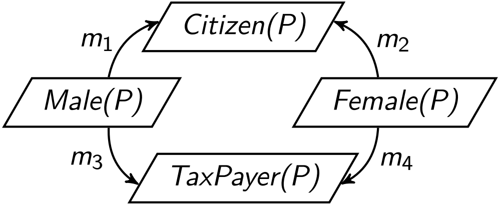
$\Rightarrow$ pure LAV and GAV mappings

- combine methods for answering queries in these settings:
  - unfolding
  - algorithms for answering queries using views
- build a rule/goal tree
- derive UCQ over $\mathcal{S}$ from it

$\Rightarrow$ sound, and for polynomial cases also complete

Query Answering: First Order Reasoning

query answering: FO reasoning over $\mathcal{P}$

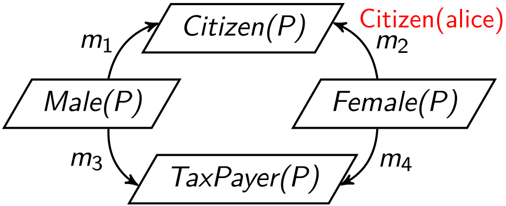


m_1: Citizen(x) :- Male(x)
m_2: Citizen(x) :- Female(x)
m_3: Male(x) $\subseteq$ TaxPayer(x)
m_4: Female(x) $\subseteq$ TaxPayer(x)

m_1, m_2:
Citizen(x) $\rightarrow$ Male(x) $\vee$ Female(x)
m_3: Male(x) $\rightarrow$ TaxPayer(x)
m_4: Female(x) $\rightarrow$ TaxPayer(x)

Query Answering: First Order Reasoning

query answering: FO reasoning over $\mathcal{P}$



m_1: Citizen(x) :- Male(x)
m_2: Citizen(x) :- Female(x)
m_3: Male(x) $\subseteq$ TaxPayer(x)
m_4: Female(x) $\subseteq$ TaxPayer(x)

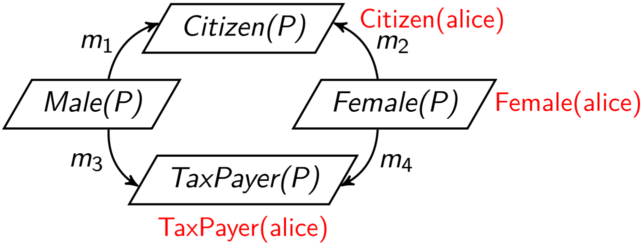
m_1, m_2:
Citizen(x) $\rightarrow$ Male(x) $\vee$ Female(x)
m_3: Male(x) $\rightarrow$ TaxPayer(x)
m_4: Female(x) $\rightarrow$ TaxPayer(x)

Example

Consider Query $\{x \mid TaxPayer(x)\}$

Query Answering: First Order Reasoning

query answering: FO reasoning over $\mathcal{P}$



m_1: Citizen(x) :- Male(x)
m_2: Citizen(x) :- Female(x)
m_3: Male(x) $\subseteq$ TaxPayer(x)
m_4: Female(x) $\subseteq$ TaxPayer(x)

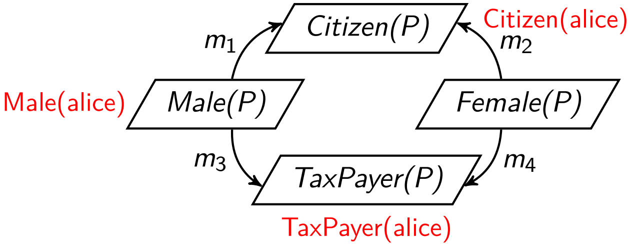
m_1, m_2:
Citizen(x) $\rightarrow$ Male(x) $\vee$ Female(x)
m_3: Male(x) $\rightarrow$ TaxPayer(x)
m_4: Female(x) $\rightarrow$ TaxPayer(x)

Example

Consider Query $\{x \mid TaxPayer(x)\}$

Query Answering: First Order Reasoning

query answering: FO reasoning over $\mathcal{P}$



m_1: Citizen(x) :- Male(x)
m_2: Citizen(x) :- Female(x)
m_3: Male(x) $\subseteq$ TaxPayer(x)
m_4: Female(x) $\subseteq$ TaxPayer(x)

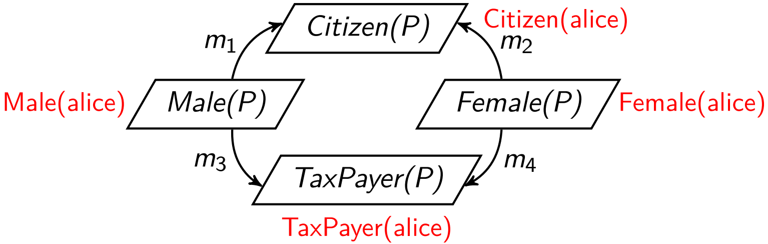
m_1, m_2:
Citizen(x) $\rightarrow$ Male(x) $\vee$ Female(x)
m_3: Male(x) $\rightarrow$ TaxPayer(x)
m_4: Female(x) $\rightarrow$ TaxPayer(x)

Example

Consider Query $\{x \mid TaxPayer(x)\}$

Query Answering: First Order Reasoning

query answering: FO reasoning over $\mathcal{P}$



m_1: Citizen(x) :- Male(x)
m_2: Citizen(x) :- Female(x)
m_3: Male(x) $\subseteq$ TaxPayer(x)
m_4: Female(x) $\subseteq$ TaxPayer(x)

m_1, m_2:
Citizen(x) $\rightarrow$ Male(x) $\vee$ Female(x)
m_3: Male(x) $\rightarrow$ TaxPayer(x)
m_4: Female(x) $\rightarrow$ TaxPayer(x)

Example

Consider Query $\{x \mid TaxPayer(x)\}$

Epistemic Logic

- A modal logic used for modeling knowledge, certainty
- Modal logic is used e.g. in multi agent systems

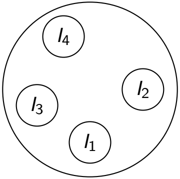
More precisely: KT45 (or S5)

Epistemic Logic

- A modal logic used for modeling knowledge, certainty
- Modal logic is used e.g. in multi agent systems

More precisely: KT45 (or S5)

- Syntax: FOL, but also $\mathbf{K}\phi$ is an atom (if ϕ is a formula)
- Semantics:
  - Often defined using Kripke structures (W, R, V)
  - Here: every world is accessible from every world
  - epistemic interpretation $\varepsilon = (I, W)$
    - W ...set of FO interpretations, $I \in W$

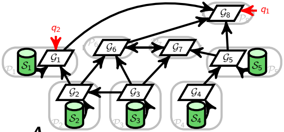


$a(\vec{x})$ satisfied in ε: by $\vec{t}$ s.t. $a(\vec{t})$ is true in I

$\mathbf{K}\phi(\vec{x})$ satisfied in ε: by $\vec{t}$ s.t.
$\phi(\vec{t})$ is satisfied in all $\varepsilon' = (J, W)$ with $J \in W$

epistemic model: ϕ is satisfied in every (J, W) $(J \in W)$

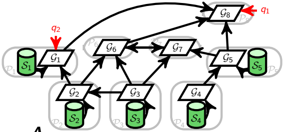
Modeling PDM [Calvanese et al., 2004]



Peer schema:

- $\mathcal{G}$ may contain function free FO formulas over $A_{\mathcal{G}}$

Modeling PDM [Calvanese et al., 2004]



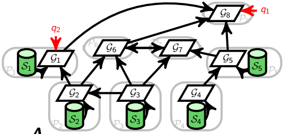
Peer schema:

- $\mathcal{G}$ may contain function free FO formulas over $A_{\mathcal{G}}$

Epistemic Theory:

- $T_{\mathcal{P}}$:
  - formulas in $\mathcal{G}$
  - $\forall \vec{x}\big(\exists \vec{y}(\phi_{\mathcal{S}}(\vec{x}, \vec{y})) \rightarrow \exists \vec{z} \psi_{\mathcal{G}}(\vec{x}, \vec{z})\big)$ (for each $m \in \mathcal{L}$)
- $M_{\mathcal{P}}$:
  - axioms $\forall \vec{x}\big(\mathbf{K}(\exists \vec{y} \phi(\vec{x}, \vec{y})) \rightarrow \exists \vec{z} \psi(\vec{x}, \vec{z})\big)$ (for each $m \in \mathcal{M}$)

Modeling PDM [Calvanese et al., 2004]



Peer schema:

- $\mathcal{G}$ may contain function free FO formulas over $A_{\mathcal{G}}$

Epistemic Theory:

- $T_{\mathcal{P}}$:
  - formulas in $\mathcal{G}$
  - $\forall \vec{x}(\exists \vec{y}(\phi_{\mathcal{S}}(\vec{x}, \vec{y})) \rightarrow \exists \vec{z} \psi_{\mathcal{G}}(\vec{x}, \vec{z}))$ (for each $m \in \mathcal{L}$)
- $M_{\mathcal{P}}$:
  - axioms $\forall \vec{x}(\mathbf{K}(\exists \vec{y} \phi(\vec{x}, \vec{y})) \rightarrow \exists \vec{z} \psi(\vec{x}, \vec{z}))$ (for each $m \in \mathcal{M}$)

Semantics:

- Recall: FOL model of $T_{\mathcal{P}}$ based on D
- Epistemic model of $\mathcal{P}$ based on D: (I, W)
  - W: set of models of $T_{\mathcal{P}}$ based on D
  - (I, W): epistemic model of $M_{\mathcal{P}}$
- Certain answers w.r.t. D: $\bigcap q^I$ for all epistemic models (I, W)

Properties of Epistemic Logic Based Semantics

(denote certain answers w.r.t. source instance D as $ans(q, \mathcal{P}, D)$)

- sound approximation of FOL: $ans_{\mathbf{K}}(q, \mathcal{P}, D) \subseteq ans_{fol}(q, \mathcal{P}, D)$

Unique Maximal Epistemic Model for $\mathcal{P}$

- (I, W) s.t. there exists no model (J, W') with $W \subset W'$
  - Unique, Independent of I

$\Rightarrow ans_{\mathbf{K}}(q, \mathcal{P}, D) = \{\vec{t} \mid \vec{t} \in q^I \text{ for each } I \in W\}$

Properties of Epistemic Logic Based Semantics

(denote certain answers w.r.t. source instance D as $ans(q, \mathcal{P}, D)$)

- sound approximation of FOL: $ans_{\mathbf{K}}(q, \mathcal{P}, D) \subseteq ans_{fol}(q, \mathcal{P}, D)$

Unique Maximal Epistemic Model for $\mathcal{P}$

- (I, W) s.t. there exists no model (J, W') with $W \subset W'$
  - Unique, Independent of I

$\Rightarrow ans_{\mathbf{K}}(q, \mathcal{P}, D) = \{\vec{t} \mid \vec{t} \in q^I \text{ for each } I \in W\}$

- $FOE(\mathcal{P}, D)$: minimal FO theory containing $T_{\mathcal{P}}$, D, and
  - for each $cq' \rightsquigarrow cq$, if $FOE(\mathcal{P}, D) \models \exists \vec{y} body_{cq'}(\vec{t}, \vec{y})$, then $\exists \vec{z} body_{cq}(\vec{t}, \vec{z}) \in FOE(\mathcal{P}, D)$

Theorem (Calvanese et al., 2004)

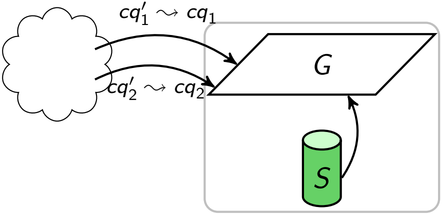
The set of interpretations $\{I \mid I \models FOE(\mathcal{P}, D)\}$ is the unique maximal epistemic model W for $\mathcal{P}$ based on D.

Intuition: Only exchange certain answers

Definition ($\tau(\mathcal{P})$)

Given $\mathcal{P}_i = (\mathcal{G}, \mathcal{S}, \mathcal{L}, \mathcal{M})$, define $\tau(\mathcal{P}_i) = (\mathcal{G}, \mathcal{S}', \mathcal{L}', \mathcal{M})$ where

1. $\mathcal{S}' = \mathcal{S} \cup \{r \mid cq' \rightsquigarrow cq \in \mathcal{M}\}$
2. $\mathcal{L}' = \mathcal{L} \cup \{\{\vec{x} \mid r(\vec{x})\} \rightsquigarrow cq \mid cq' \rightsquigarrow cq \in \mathcal{M}\}$

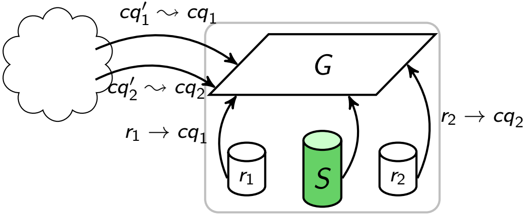


Intuition: Only exchange certain answers

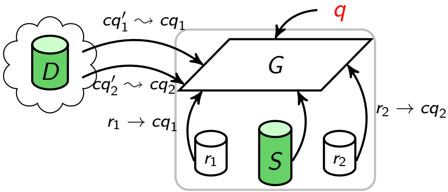
Definition ($\tau(\mathcal{P})$)

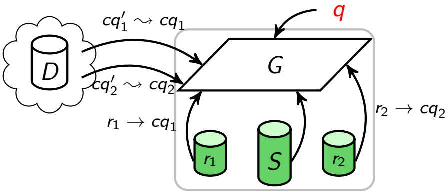
Given $\mathcal{P}_i = (\mathcal{G}, \mathcal{S}, \mathcal{L}, \mathcal{M})$, define $\tau(\mathcal{P}_i) = (\mathcal{G}, \mathcal{S}', \mathcal{L}', \mathcal{M})$ where

1. $\mathcal{S}' = \mathcal{S} \cup \{r \mid cq' \rightsquigarrow cq \in \mathcal{M}\}$
2. $\mathcal{L}' = \mathcal{L} \cup \{\{\vec{x} \mid r(\vec{x})\} \rightsquigarrow cq \mid cq' \rightsquigarrow cq \in \mathcal{M}\}$



Intuition: Only exchange certain answers (contd.)





Given $\mathcal{P}$, $\mathcal{P}_i = (\mathcal{G}, \mathcal{S}, \mathcal{L}, \mathcal{M})$, D and query q over $\mathcal{G}$:

- Let $\bar{D}$ be source instance for $\tau(\mathcal{P}_i)$ s.t.
  - $\mathcal{S}^{\bar{D}} = \mathcal{S}^D$ and $r^{\bar{D}} = ans(cq', \mathcal{P}, D)$
- We want $ans(q, \mathcal{P}, D) = ans(q, \tau(\mathcal{P}), \bar{D})$

provides: modularity and independence

Intuition: Only exchange certain answers (contd.)

Recall intuition: $ans(q, \mathcal{P}, D) = ans(q, \tau(\mathcal{P}), \bar{D})$

- for $cq' \rightsquigarrow cq \in \mathcal{M}$:
  - $r \in S'$, $\{\vec{x} \mid r(\vec{x})\} \rightsquigarrow cq \in \mathcal{L}'$
  - $r^{\bar{D}} = ans(cq', \mathcal{P}, D)$

Intuition: Only exchange certain answers (contd.)

Recall intuition: $ans(q, \mathcal{P}, D) = ans(q, \tau(\mathcal{P}), \bar{D})$

- for $cq' \rightsquigarrow cq \in \mathcal{M}$:
  - $r \in S'$, $\{\vec{x} \mid r(\vec{x})\} \rightsquigarrow cq \in \mathcal{L}'$
  - $r^{\bar{D}} = ans(cq', \mathcal{P}, D)$

Further recall:

- $ans_{\mathbf{K}}(cq', \mathcal{P}, D) = \{\vec{t} \mid \vec{t} \in cq'^{I}$ for each $I \in W\}$
  - for W: maximal epistemic model
- axiom $\forall\vec{x}(\mathbf{K}(\exists\vec{y}\, body_{cq'}(\vec{x},\vec{y})) \rightarrow \exists\vec{z}\, body_{cq}(\vec{x},\vec{z}))$

Intuition: Only exchange certain answers (contd.)

Recall intuition: $ans(q, \mathcal{P}, D) = ans(q, \tau(\mathcal{P}), \bar{D})$

- for $cq' \rightsquigarrow cq \in \mathcal{M}$:
  - $r \in S'$, $\{\vec{x} \mid r(\vec{x})\} \rightsquigarrow cq \in \mathcal{L}'$
  - $r^{\bar{D}} = ans(cq', \mathcal{P}, D)$

Further recall:

- $ans_{\mathbf{K}}(cq', \mathcal{P}, D) = \{\vec{t} \mid \vec{t} \in cq'^{I} \text{ for each } I \in W\}$
  - for W: maximal epistemic model
- axiom $\forall\vec{x}(\mathbf{K}(\exists\vec{y}\,body_{cq'}(\vec{x},\vec{y})) \rightarrow \exists\vec{z}\,body_{cq}(\vec{x},\vec{z}))$

Hence (informal)

- $ans_{\mathbf{K}}(cq', \mathcal{P}, D) = \{\vec{t} \mid \text{for each } I \in W: \exists\vec{y}\text{: } body_{cq'}(\vec{t},\vec{y}) \in I\}$
- $\mathbf{K}(\exists\vec{y}\,body_{cq'}(\vec{x},\vec{y}))$ satisfied by tuples
  $\{\vec{t} \mid \text{in each } I \in W: \exists\vec{y}\text{: } body_{cq'}(\vec{t},\vec{y}) \in I\}$

$\Rightarrow \mathcal{P}$ "imports" the same tuples

Query Answering

use this idea for query answering $\Rightarrow$ always consider $\tau(\mathcal{P})$

perfect reformulation

Given query q over $\mathcal{G}_i$ $\Rightarrow$ query q_1 over $\mathcal{S}_i'$ s.t. for every instance D_1 for $\tau(\mathcal{P})$, $q_1^{D_1} = ans(q, \tau(\mathcal{P}), D_1)$

(assume settings where perfect reformulation always exists)

Query Answering

use this idea for query answering $\Rightarrow$ always consider $\tau(\mathcal{P})$

perfect reformulation

Given query q over $\mathcal{G}_i \Rightarrow$ query q_1 over $\mathcal{S}'_i$ s.t. for every instance D_1 for $\tau(\mathcal{P})$, $q_1^{D_1} = ans(q, \tau(\mathcal{P}), D_1)$

(assume settings where perfect reformulation always exists)

Idea of the Algorithm

- Compute a datalog program DP, containing
  - facts from $\mathcal{S}$
  - rules encoding perfect reformulations to $\mathcal{S}'$

Query Answering

use this idea for query answering $\Rightarrow$ always consider $\tau(\mathcal{P})$

perfect reformulation

Given query q over $\mathcal{G}_i$ $\Rightarrow$ query q_1 over $\mathcal{S}'_i$ s.t. for every instance D_1 for $\tau(\mathcal{P})$, $q_1^{D_1} = ans(q, \tau(\mathcal{P}), D_1)$

(assume settings where perfect reformulation always exists)

Idea of the Algorithm

- Compute a datalog program DP, containing
  - facts from $\mathcal{S}$
  - rules encoding perfect reformulations to $\mathcal{S}'$

Theorem (Calvanese et al., 2004)

1. *$Eval(head_q, DP)$ computes $ans_{\mathbf{K}}(q, \mathcal{P}, D)$*
2. *Given $\mathcal{P}$, q, $\vec{t}$, deciding $\vec{t} \in ans_{\mathbf{K}}(q, \mathcal{P}, D)$ is PTIME-complete (data complexity)*

Query Answering: Algorithm

query answering algorithm

at $\mathcal{P}_i$: peerQueryHandler(q,r_q)

(1) DP_I = computePerfectRef(q,r_q,$\mathcal{P}_i$); $DP_E = \emptyset$

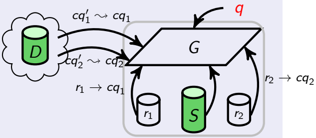
(2) for each $r \in \mathcal{S}'_i \cap DP_I$:

(3) if $r \in \mathcal{S}$ (*)

(3a) $DP_E = DP_E \cup \{r(\vec{t}) \mid r(\vec{t}) \in D\}$

else ($r \in \mathcal{S}' \setminus \mathcal{S}$)

(3b) $DP' = \mathcal{P}'$.peerQueryHandler($Q(r)$,r)
$DP_I = DP_I \cup DP'_I$; $DP_E = DP_E \cup DP'_E$



(4) return DP

(*) loop detection omitted

Further nice properties

Decidability depends only on local properties

- under FOL: also constraints may be propagated by mappings
- Epistemic Logic: provides complete modularity for peers

Mapping Composition

- Semantics allows for (reasonable) mapping composition
- Resulting systems are query equivalent

Inconsistency Handling

- Consider two kinds of inconsistency:
  - local inconsistency, P2P inconsistency
- Use nonmonotonic extension ($K45_n^A$), model $cq_i \rightsquigarrow cq_j$:
  - $\forall \vec{x}\big(\neg \mathbf{A}_i \bot_i \wedge \mathbf{K}_i(\exists \vec{y}\, body_{cq_i}(\vec{x},\vec{y})) \wedge \neg \mathbf{A}_j(\neg \exists \vec{z}\, body_{cq_j}(\vec{x},\vec{z})) \rightarrow \mathbf{K}_j(\exists \vec{z}\, body_{cq_j}(\vec{x},\vec{z}))\big)$

Outline

Idea

So far: Peer Data Integration

- data remains local at peers
- information needed for query answering are exchanged
- mappings can be considered as “virtual”

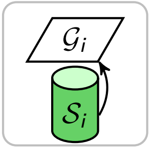
Other possibility: Generalize Data Exchange

- copy data between different peers
- interpret mappings as constraints
- materialize data to satisfy these constraints

$\rightarrow$ Look onto some approaches following this idea

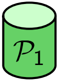
Reconciling Data Exchange and PDM

[De Giacomo et al., PODS 2007]



Reconciling Data Exchange and PDM

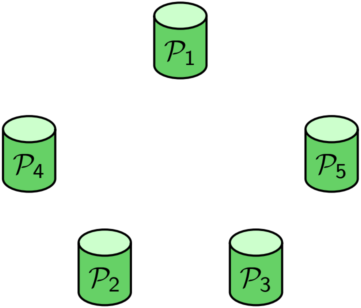
[De Giacomo et al., PODS 2007]



Reconciling Data Exchange and PDM

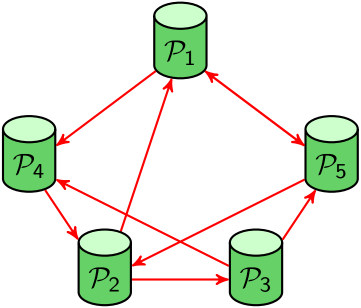
[De Giacomo et al., PODS 2007]

$$\mathcal{S} = \langle \mathcal{P}, \quad , \quad \rangle$$



Reconciling Data Exchange and PDM

[De Giacomo et al., PODS 2007]

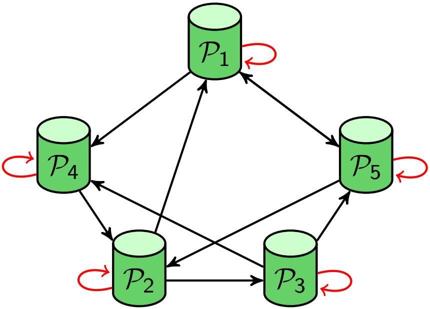


$$\mathcal{S} = \langle \mathcal{P}, \quad , \mathcal{M}_E \rangle$$

- $\mathcal{M}_E$: TGDs between pairs of peers

Reconciling Data Exchange and PDM

[De Giacomo et al., PODS 2007]

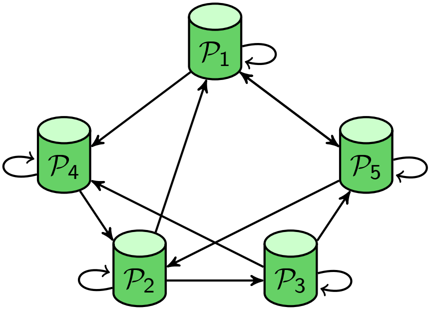


$$\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E \rangle$$

- $\mathcal{M}_E$: TGDs between pairs of peers
- $\mathcal{C}_E$: TGDs & EGDs over single peer

Reconciling Data Exchange and PDM

[De Giacomo et al., PODS 2007]



$$\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E \rangle$$

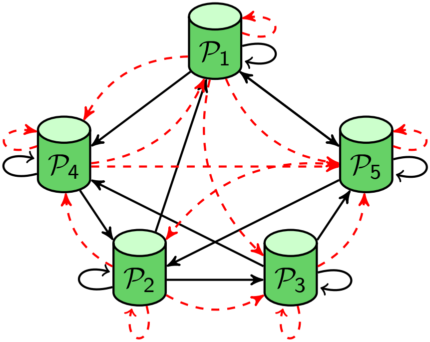
- $\mathcal{M}_E$: TGDs between pairs of peers
- $\mathcal{C}_E$: TGDs & EGDs over single peer

Semantics:

- $\mathcal{C}_E$: FO semantics
- $\mathcal{M}_E$: exchanges only certain answers
- Universal $\mathcal{S}$-solution

Reconciling Data Exchange and PDM

[De Giacomo et al., PODS 2007]

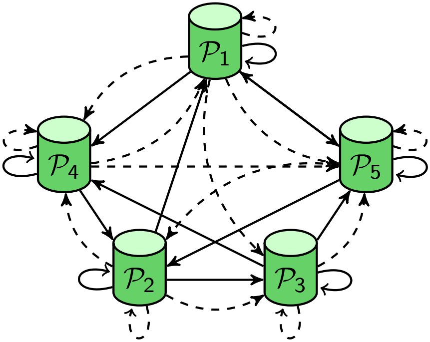


$\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$

- $\mathcal{M}_E$, $\mathcal{M}_I$: TGDs between pairs of peers
- $\mathcal{C}_E$, $\mathcal{C}_I$: TGDs & EGDs over single peer

Reconciling Data Exchange and PDM

[De Giacomo et al., PODS 2007]



$$\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$$

- $\mathcal{M}_E$, $\mathcal{M}_I$: TGDs between pairs of peers
- $\mathcal{C}_E$, $\mathcal{C}_I$: TGDs & EGDs over single peer

Semantics:

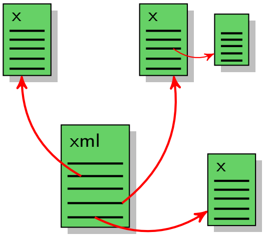
- $\mathcal{C}_E$, $\mathcal{C}_I$: FO semantics
- $\mathcal{M}_E$, $\mathcal{M}_I$: certain answers
- $\mathcal{C}_I$, $\mathcal{M}_I$ precedence
- Universal $\mathcal{S}$-solution

Active XML

Active XML

Active XML

Recall: Active XML



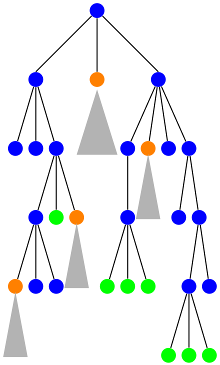
Not considered yet

- Formal semantics
  - service call
  - query answering
- Complexity

[Abiteboul et al., PODS 2004]

→ consider only **monotone Web Services**

AXML Document



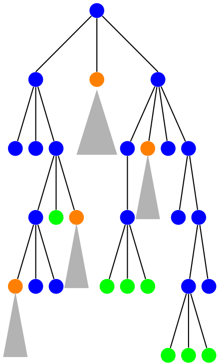
Definition (AXML document)

AXML document: pair (T, λ) where

- $T = (N, E)$: finite, unordered tree
  - $N \subset \mathcal{N}$: finite set of nodes
  - $E \subset N \times N$: directed edges
- $\lambda\colon N \to \mathcal{L} \cup \mathcal{F} \cup \mathcal{V}$: function s.t.
  - $\lambda(n) \in \mathcal{V}$ only if n is a leaf node
  - for root n, $\lambda(n) \in \mathcal{V} \cup \mathcal{L}$

$\mathcal{D}$: document names, $\mathcal{N}$: nodes, $\mathcal{L}$: labels,
$\mathcal{V}$: atomic values, $\mathcal{F}$: function names

AXML Document



data nodes, function nodes

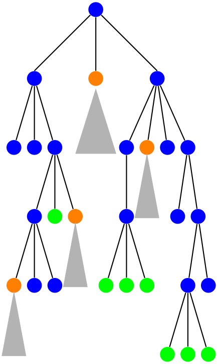
Definition (AXML document)

AXML document: pair (T, λ) where

- $T = (N, E)$: finite, unordered tree
  - $N \subset \mathcal{N}$: finite set of nodes
  - $E \subset N \times N$: directed edges
- $\lambda \colon N \to \mathcal{L} \cup \mathcal{F} \cup \mathcal{V}$: function s.t.
  - $\lambda(n) \in \mathcal{V}$ only if n is a leaf node
  - for root n, $\lambda(n) \in \mathcal{V} \cup \mathcal{L}$

$\mathcal{D}$: document names, $\mathcal{N}$: nodes, $\mathcal{L}$: labels, $\mathcal{V}$: atomic values, $\mathcal{F}$: function names

AXML Document



data nodes, function nodes

Definition (AXML document)

AXML document: pair (T, λ) where

- $T = (N, E)$: finite, unordered tree
  - $N \subset \mathcal{N}$: finite set of nodes
  - $E \subset N \times N$: directed edges
- $\lambda \colon N \to \mathcal{L} \cup \mathcal{F} \cup \mathcal{V}$: function s.t.
  - $\lambda(n) \in \mathcal{V}$ only if n is a leaf node
  - for root n, $\lambda(n) \in \mathcal{V} \cup \mathcal{L}$

$\mathcal{D}$: document names, $\mathcal{N}$: nodes, $\mathcal{L}$: labels, $\mathcal{V}$: atomic values, $\mathcal{F}$: function names

function call: pass subtree as parameter; get forest as return value
$\Rightarrow$ append as siblings to call node

Reduced Documents

Definition

- (T_1, λ_1) is subsumed by (T_2, λ_2) ($(T_1, \lambda_1) \subseteq (T_2, \lambda_2)$) if there exists mapping $h \colon N_1 \to N_2$ s.t:
  - $h(root(T_1)) = root(T_2)$
  - n_1 child of $n_2 \Rightarrow h(n_1)$ child of $h(n_2)$ (for all $n_1, n_2 \in N_1$)
  - $\lambda_1(n) = \lambda_2(h(n))$ (for all $n \in N_1$)
- $d_1 \subseteq d_2$ and $d_2 \subseteq d_1 \Rightarrow d_1 \equiv d_2$

$\rightarrow$ Document d is reduced if for no subtree d' of d, $d \equiv d'$

Reduced Documents

Definition

- (T_1, λ_1) is subsumed by (T_2, λ_2) ($(T_1, \lambda_1) \subseteq (T_2, \lambda_2)$) if there exists mapping $h \colon N_1 \to N_2$ s.t:
  - $h(root(T_1)) = root(T_2)$
  - n_1 child of $n_2 \Rightarrow h(n_1)$ child of $h(n_2)$ (for all $n_1, n_2 \in N_1$)
  - $\lambda_1(n) = \lambda_2(h(n))$ (for all $n \in N_1$)
- $d_1 \subseteq d_2$ and $d_2 \subseteq d_1 \Rightarrow d_1 \equiv d_2$

$\to$ Document d is reduced if for no subtree d' of d, $d \equiv d'$

- Properties:
  - Each document has a unique reduced version
  - Decision and Function problem solvable in PTIME

Monotone AXML Systems

Definition (monotone AXML system)

- monotone AXML system: $\mathcal{S} = (D, F, I)$
  - finite sets $D \subset \mathcal{D}$, $F \subset \mathcal{F}$
  - mapping I: for $d \in D$, $I(d)$ returns a document,
    for $f \in F$, $I(f)$ returns a monotone service

Monotone AXML Systems

Definition (monotone AXML system)

- monotone AXML system: $\mathcal{S} = (D, F, I)$
  - finite sets $D \subset \mathcal{D}$, $F \subset \mathcal{F}$
  - mapping I: for $d \in D$, $I(d)$ returns a document,
    for $f \in F$, $I(f)$ returns a monotone service

- (web) service s
  - defined w.r.t. set $\{d_1, \ldots, d_n\}$ of document names
  - given assignment θ of AXML documents to $\{d_1, \ldots, d_n\}$, return forest of AXML documents
  - consider s as black box

- monotone service
  - for all θ, θ': for all i: $\theta(d_i) \subseteq \theta'(d_i) \Rightarrow s(\theta) \subseteq s(\theta')$

Invocations of Services

- Service invocation
  - given $\mathcal{S}$, $d \in D$, $v \in I(d)$, $\lambda(v) = f$
  - invoking f: call $I(f)$ on θ: $\theta(d_i) = I(d_i)$, $\theta(input)$, $\theta(context)$
  - append $I(f)(\theta)$ to parent of v, normalize afterward

Invocations of Services

- Service invocation
  - given $\mathcal{S}$, $d \in D$, $v \in I(d)$, $\lambda(v) = f$
  - invoking f: call $I(f)$ on θ: $\theta(d_i) = I(d_i)$, $\theta(input)$, $\theta(context)$
  - append $I(f)(\theta)$ to parent of v, normalize afterward

- Sequences of Invocations
  - $\mathcal{S} \xrightarrow{v} \mathcal{S}'$: $\mathcal{S}' \not\equiv \mathcal{S}$; $\mathcal{S}'$ obtained from $\mathcal{S}$ by invoking function at node v
  - rewriting (possible infinite): $\mathcal{S} \xrightarrow{v_1} \mathcal{S}_1 \xrightarrow{v_2} \mathcal{S}_2 \rightarrow \ldots \xrightarrow{v_n} \mathcal{S}_n \ldots$ $(\mathcal{S} \xrightarrow{*} \mathcal{S}_n)$
  - system terminates in $\mathcal{S}_n$: no $v_{n+1}, \mathcal{S}_{n+1}$ s.t. $\mathcal{S}_n \xrightarrow{v_{n+1}} \mathcal{S}_{n+1}$

Invocations of Services

- Service invocation
  - given $\mathcal{S}$, $d \in D$, $v \in I(d)$, $\lambda(v) = f$
  - invoking f: call $I(f)$ on θ: $\theta(d_i) = I(d_i)$, $\theta(input)$, $\theta(context)$
  - append $I(f)(\theta)$ to parent of v, normalize afterward

- Sequences of Invocations
  - $\mathcal{S} \xrightarrow{v} \mathcal{S}'$: $\mathcal{S}' \not\equiv \mathcal{S}$; $\mathcal{S}'$ obtained from $\mathcal{S}$ by invoking function at node v
  - rewriting (possible infinite): $\mathcal{S} \xrightarrow{v_1} \mathcal{S}_1 \xrightarrow{v_2} \mathcal{S}_2 \rightarrow \ldots \xrightarrow{v_n} \mathcal{S}_n \ldots$ $(\mathcal{S} \xrightarrow{*} \mathcal{S}_n)$
  - system terminates in $\mathcal{S}_n$: no $v_{n+1}, \mathcal{S}_{n+1}$ s.t. $\mathcal{S}_n \xrightarrow{v_{n+1}} \mathcal{S}_{n+1}$

- fair (infinite) sequence
  - for every $v_i \in \mathcal{S}_i$: there exists a $j > i$ s.t. either $\mathcal{S}_j \xrightarrow{v_i} \mathcal{S}_{j+1}$ or invoking v_i has no effect on $\mathcal{S}_j$

Semantics of monotone AXML systems

Definition (semantics of monotone AXML systems)

For a monotone AXML system $\mathcal{S}$, its semantics $[\mathcal{S}]$ is defined as:

- $[\mathcal{S}] = \mathcal{J}$ if $\mathcal{S} \xrightarrow{*} \mathcal{J}$ and system terminates at $\mathcal{J}$ ($\mathcal{J}$ finite)
- $[\mathcal{S}] = \bigcup \mathcal{S}_i$ for infinite fair rewriting $\mathcal{S} \ldots \rightarrow \ldots \xrightarrow{v_i} \mathcal{S}_i \ldots$

Semantics of monotone AXML systems

Definition (semantics of monotone AXML systems)

For a monotone AXML system $\mathcal{S}$, its semantics $[\mathcal{S}]$ is defined as:

- $[\mathcal{S}] = \mathcal{J}$ if $\mathcal{S} \xrightarrow{*} \mathcal{J}$ and system terminates at $\mathcal{J}$ ($\mathcal{J}$ finite)
- $[\mathcal{S}] = \bigcup \mathcal{S}_i$ for infinite fair rewriting $\mathcal{S} \ldots \rightarrow \ldots \xrightarrow{v_i} \mathcal{S}_i \ldots$

- Semantics is well defined
  (order of invocations does not matter)
  - $\mathcal{S} \xrightarrow{*} \hat{\mathcal{S}}$ and $\mathcal{S} \xrightarrow{*} \bar{\mathcal{S}}$: either $\bar{\mathcal{S}} \subseteq \mathcal{S}'$ ($\hat{\mathcal{S}}$ terminates at $\mathcal{S}'$), or $\bar{\mathcal{S}} \subseteq \mathcal{S}_i$ for some i ($\hat{\mathcal{S}}$ not terminating)
  - one rewriting terminates at $\mathcal{J} \Rightarrow$ any rewriting terminates at $\mathcal{J}$
  - one fair rewriting does not terminate $\Rightarrow$ no rewriting terminates; any fair rewriting results in same infinite system

Positive Active XML

Also consider service implementations, defined as queries

Positive Active XML

Also consider service implementations, defined as queries

Definition (Positive Query)

positive query q: r :- $d_1/p_1, \ldots, d_n/p_n, e_1, \ldots, e_m$ where

- d_i: document names, r, p_i: positive AXML tree patterns
- each variable occurring in r also occurs in some p_i
- e_j: inequalities $x \neq y$ between label, function, or value variables or constants (no tree variables).
  No tree variable occurs twice in the body

simple query: no tree variables

AXML tree pattern:

- subtree of AXML document
- some labels replaced by label variables

Query Semantics

- Recall:
  - query $q = r \text{ :- } d_1/p_1, \ldots, d_n/p_n, e_1, \ldots, e_m$
  - monotone AXML system $\mathcal{S} = (D, F, I)$

Query Semantics

- Recall:
  - query $q = r$:- $d_1/p_1, \ldots, d_n/p_n, e_1, \ldots, e_m$
  - monotone AXML system $\mathcal{S} = (D, F, I)$

- Snapshot Result $q(\mathcal{S})$
  - consider variable assignments μ (respect typing) s.t.
    - for each $d_i/p_i \in q$: $\mu(p_i) \subseteq I(d_i)$
  - $q(\mathcal{S})$: forest of all documents $\mu(r)$

Query Semantics

- Recall:
  - query $q = r \text{ :- } d_1/p_1, \ldots, d_n/p_n, e_1, \ldots, e_m$
  - monotone AXML system $\mathcal{S} = (D, F, I)$

- Snapshot Result $q(\mathcal{S})$
  - consider variable assignments μ (respect typing) s.t.
    - for each $d_i/p_i \in q$: $\mu(p_i) \subseteq I(d_i)$
  - $q(\mathcal{S})$: forest of all documents $\mu(r)$
  - Properties:
    - monotone (i.e. $\mathcal{S} \subseteq \mathcal{S}' \Rightarrow q(\mathcal{S}) \subseteq q(\mathcal{S}')$) for positive queries (no inequalities of tree variables)
    - for positive queries: PTIME

Query Semantics

- Recall:
  - query $q = r$:- $d_1/p_1, \ldots, d_n/p_n, e_1, \ldots, e_m$
  - monotone AXML system $\mathcal{S} = (D, F, I)$

- Snapshot Result $q(\mathcal{S})$
  - consider variable assignments μ (respect typing) s.t.
    - for each $d_i/p_i \in q$: $\mu(p_i) \subseteq I(d_i)$
  - $q(\mathcal{S})$: forest of all documents $\mu(r)$
  - Properties:
    - monotone (i.e. $\mathcal{S} \subseteq \mathcal{S}' \Rightarrow q(\mathcal{S}) \subseteq q(\mathcal{S}')$) for positive queries (no inequalities of tree variables)
    - for positive queries: PTIME

- Query Result $[q](\mathcal{S})$
  - $[q](\mathcal{S}) = q([\mathcal{S}])$ if $\mathcal{S}$ converges to finite system $[\mathcal{S}]$
  - $[q](\mathcal{S}) = \bigcup q(\mathcal{S}_i)$ for infinite fair rewriting $\mathcal{S} \ldots \mathcal{S}_i \ldots$ otherwise
  - for positive queries: result is independent of rewriting sequence

Positive Systems

service descriptions $I(f)$ defined as positive queries
if all queries are simple $\rightarrow$ simple positive system

Positive Systems

service descriptions $I(f)$ defined as positive queries
if all queries are simple $\rightarrow$ simple positive system

Semantics of positive systems

- positive system $\mathcal{S}$, function node v, $\lambda(v) = f$, $I(f) = q$
- invoking f: evaluate q under θ
- snapshot result of $q(\mathcal{S})$ is added as sibling of v

Positive Systems

service descriptions $I(f)$ defined as positive queries
if all queries are simple $\rightarrow$ simple positive system

Semantics of positive systems

- positive system $\mathcal{S}$, function node v, $\lambda(v) = f$, $I(f) = q$
- invoking f: evaluate q under θ
- snapshot result of $q(\mathcal{S})$ is added as sibling of v

Complexity

Theorem (Abiteboul et al., PODS 2004)

Any Turing Machine can be simulated by a positive AXML system, with the input tape represented by an AXML tree.

$\Rightarrow$ it is undecidable whether a positive system terminates

Restricted Systems

Try to find decidable systems

Acyclic Systems

- dependency graph (V, E) of $\mathcal{S} = (D, F, I)$:
  - V: $D \cup F$ (document and function names)
  - E: edge (d, f) if f occurs in $I(d)$,
    edge (f, d) (resp. (f, g)) if d (resp. g) occurs in $I(f)$
- AXML system acyclic if dependency graph is acyclic
- acyclic systems always terminate

Restricted Systems

Try to find decidable systems

Acyclic Systems

- dependency graph (V, E) of $\mathcal{S} = (D, F, I)$:
  - V: $D \cup F$ (document and function names)
  - E: edge (d, f) if f occurs in $I(d)$,
    edge (f, d) (resp. (f, g)) if d (resp. g) occurs in $I(f)$
- AXML system acyclic if dependency graph is acyclic
- acyclic systems always terminate

Simple Positive Systems

- Recall: simple queries: no tree variables
- For every simple positive system $\mathcal{S}$:
  - $[\mathcal{S}]$ is regular
  - compute finite graph representation of $[\mathcal{S}]$ in $\mathrm{EXPTIME}$
  - termination: decidable in $\mathrm{EXPTIME}$, coNP hard

Querying Positive Systems

Instead of materialization: just consider query answering

Definition (q-finite)

AXML system $\mathcal{S}$ is q-finite if $[q](\mathcal{S})$ is finite

Querying Positive Systems

Instead of materialization: just consider query answering

Definition (q-finite)

AXML system $\mathcal{S}$ is q-finite if $[q](\mathcal{S})$ is finite

q: non-simple query

- undecidable whether positive system $\mathcal{S}$ is q-finite
- acyclic systems are q-finite
- simple positive systems: deciding q-finiteness is coNP hard and in $\mathrm{EXPTIME}$

Querying Positive Systems

Instead of materialization: just consider query answering

Definition (q-finite)

AXML system $\mathcal{S}$ is q-finite if $[q](\mathcal{S})$ is finite

q: non-simple query

- undecidable whether positive system $\mathcal{S}$ is q-finite
- acyclic systems are q-finite
- simple positive systems: deciding q-finiteness is coNP hard and in EXPTIME

q: simple query

- result is always finite
- BUT: for non-simple positive systems $\mathcal{S}$:
  testing if $[q](\mathcal{S})$ is nonempty is undecidable

Lazy Query Evaluation

It might not be necessary to invoke a service answering a query

- irrelevant for answer
- just return call to service in answer (lazy evaluation)

Lazy Query Evaluation

It might not be necessary to invoke a service answering a query

- irrelevant for answer
- just return call to service in answer (lazy evaluation)

Definition (possible answer)

AXML document α is a possible answer if $[\alpha] = [[q](I)]$

Lazy Query Evaluation

It might not be necessary to invoke a service answering a query

- irrelevant for answer
- just return call to service in answer (lazy evaluation)

Definition (possible answer)

AXML document α is a possible answer if $[\alpha] = [[q](I)]$

$\Rightarrow$ not expanding function nodes N still gives a possible answer?
(q–unneeded)

Lazy Query Evaluation

It might not be necessary to invoke a service answering a query

- irrelevant for answer
- just return call to service in answer (lazy evaluation)

Definition (possible answer)

AXML document α is a possible answer if $[\alpha] = [[q](I)]$

$\Rightarrow$ not expanding function nodes N still gives a possible answer?

(q–unneeded)

- Given positive AXML system $\mathcal{S}$, q, N in $\mathcal{S}$, t:
  - undecidable if: d is possible answer to q; function nodes in N need not be expanded; no more function needs to be expanded
  - For simple systems: in $\mathrm{NEXPTIME}$, coNP hard

Updates in PDM

Updates in Peer Data Management

Updates in PDM

Updates in Peer Data Management

Updates

- in PDI: no problem
- in PDM: may lead to inconsistencies $\Rightarrow$ problem

Updates in PDM

Updates in Peer Data Management

Updates

- in PDI: no problem
- in PDM: may lead to inconsistencies $\Rightarrow$ problem

Other concerns

- so far: “global” systems
- Trust
- Provenance information

Updates in PDM

Updates in Peer Data Management

Updates

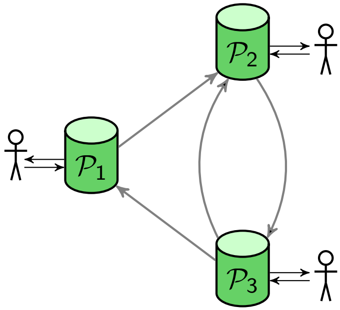
- in PDI: no problem
- in PDM: may lead to inconsistencies $\Rightarrow$ problem

Other concerns

- so far: "global" systems
- Trust
- Provenance information

Take a look onto the ORCHESTRA system

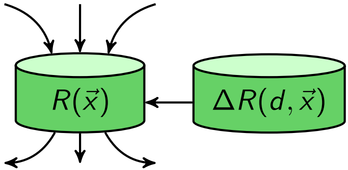
General Setting



- schema mappings:
  - (weakly acyclic) sets of TGDs
- users work on their local copies
- from time to time, they
  - publish their updates and
  - retrieve updates of other users
- trust conditions on the mappings

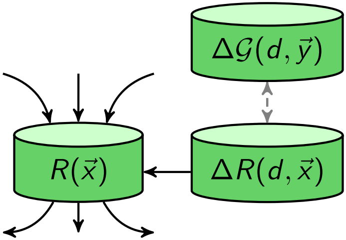
$\Rightarrow$ need for provenance information

Update Propagation



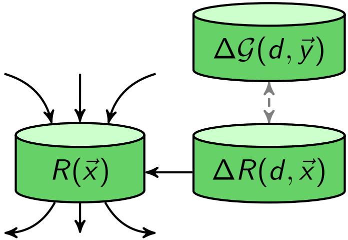
- User Actions:
  - Insert, Delete, Publish/Import
- Maintain local edit log
- Answers over local database
  - consistent with local edit log
  - for imported updates: certain answers

Update Propagation



- User Actions:
  - Insert, Delete, Publish/Import
- Maintain local edit log
- Answers over local database
  - consistent with local edit log
  - for imported updates: certain answers

Update Propagation



- User Actions:
  - Insert, Delete, Publish/Import
- Maintain local edit log
- Answers over local database
  - consistent with local edit log
  - for imported updates: certain answers

$\Rightarrow$ **what data to materialize**

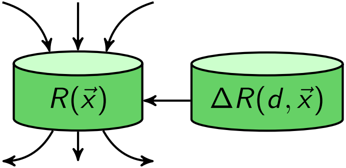
inconsistent updates:

- reconciliation algorithm (Taylor, Ives; Sigmod 2006)
  - resolve conflicts using priority mappings
  - user interaction if merging not possible

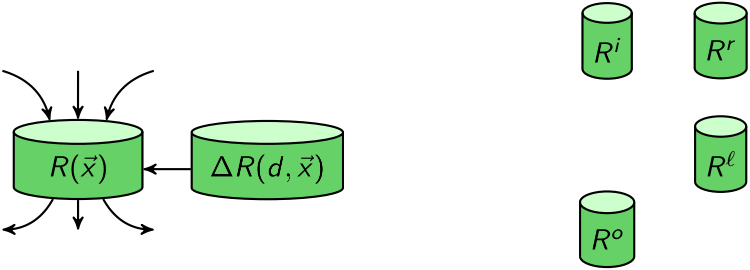
here: assume consistent updates

concentrate on what data to materialize

Semantics of Update Exchange

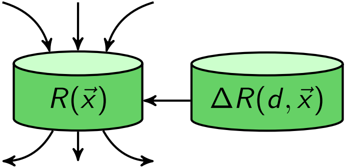


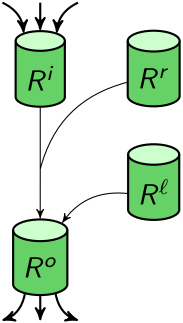
Semantics of Update Exchange



- Split every relation R:
  - R^ℓ: local contributions table
  - R^r: rejections table
  - R^i: input table
  - R^o: output table

Semantics of Update Exchange

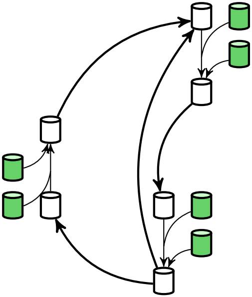




- Split every relation R:
  - R^ℓ: local contributions table
  - R^r: rejections table
  - R^i: input table
  - R^o: output table

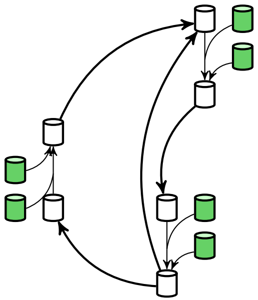
- Translate mappings $\Sigma \rightarrow \Sigma'$:
  - for each $m \in \mathcal{M}$: replace R
    - in lhs by R^o and
    - in rhs by R^i
  - $R^i(\vec{x}) \wedge \neg R^r(\vec{x}) \rightarrow R^o(\vec{x})$
  - $R^\ell(\vec{x}) \rightarrow R^o(\vec{x})$

Semantics of Update Exchange (contd.)



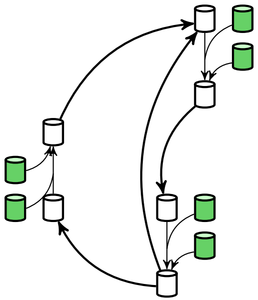
- Recall Σ':
  - $R^i(\vec{x}) \wedge \neg R^r(\vec{x}) \rightarrow R^o(\vec{x})$
  - $R^\ell(\vec{x}) \rightarrow R^o(\vec{x})$
  - $\mathcal{M}'$: weakly acyclic TGDs

Semantics of Update Exchange (contd.)



- Recall Σ':
  - $R^i(\vec{x}) \land \neg R^r(\vec{x}) \rightarrow R^o(\vec{x})$
  - $R^\ell(\vec{x}) \rightarrow R^o(\vec{x})$
  - $\mathcal{M}'$: weakly acyclic TGDs
- Publish:
  - create new instance of R^r, R^ℓ
- Import:
  - recompute R^i, R^o (chase)

Semantics of Update Exchange (contd.)



- Recall Σ':
  - $R^i(\vec{x}) \wedge \neg R^r(\vec{x}) \rightarrow R^o(\vec{x})$
  - $R^\ell(\vec{x}) \rightarrow R^o(\vec{x})$
  - $\mathcal{M}'$: weakly acyclic TGDs
- Publish:
  - create new instance of R^r, R^ℓ
- Import:
  - recompute R^i, R^o (chase)

Definition (consistent system state)

Instance $\langle I, J\rangle$ over schema $\langle \bigcup R^\ell \cup \bigcup R^r, \bigcup R^o \cup \bigcup R^i \rangle$ is consistent if $J = chase_{\Sigma'}(I)$

computable in polynomial time (data complexity)

Provenance

Need to track from where tuples are derived, and how

Provenance Token

- base tuple: tuple id
- derived tuple: polynomial
  - binary operators $+, \cdot$
  - unary function for each mapping

Provenance

Need to track from where tuples are derived, and how

Provenance Token

- base tuple: tuple id
- derived tuple: polynomial
  - binary operators $+, \cdot$
  - unary function for each mapping

Example (Provenance Tokens)

Relations R_1, R_2, Mappings
$m_1 : R_1(A, B) \rightarrow R_2(A, B)$,
$m_2 : R_2(A, B) \wedge R_1(B, C) \rightarrow R_2(A, C)$

Provenance

Need to track from where tuples are derived, and how

Provenance Token

- base tuple: tuple id
- derived tuple: polynomial
  - binary operators $+, \cdot$
  - unary function for each mapping

Example (Provenance Tokens)

Relations R_1, R_2, Mappings
$m_1: R_1(A, B) \rightarrow R_2(A, B)$,
$m_2: R_2(A, B) \wedge R_1(B, C) \rightarrow R_2(A, C)$

$r_1: R_1(a, b)$, $r_2: R_1(b, c)$, $r_3: R_1(a, c)$
$r_4: R_1(c, d)$

Provenance

Need to track from where tuples are derived, and how

Provenance Token

- base tuple: tuple id
- derived tuple: polynomial
  - binary operators $+, \cdot$
  - unary function for each mapping

Example (Provenance Tokens)

Relations R_1, R_2, Mappings
$m_1 : R_1(A, B) \rightarrow R_2(A, B)$,
$m_2 : R_2(A, B) \wedge R_1(B, C) \rightarrow R_2(A, C)$

$r_1 : R_1(a, b)$, $r_2 : R_1(b, c)$, $r_3 : R_1(a, c)$
$r_4 : R_1(c, d)$

$Pv(R_2(a, b)) : m_1(r_1)$

Provenance

Need to track from where tuples are derived, and how

Provenance Token

- base tuple: tuple id
- derived tuple: polynomial
  - binary operators $+, \cdot$
  - unary function for each mapping

Example (Provenance Tokens)

Relations R_1, R_2, Mappings
$m_1 : R_1(A, B) \rightarrow R_2(A, B)$,
$m_2 : R_2(A, B) \wedge R_1(B, C) \rightarrow R_2(A, C)$

$r_1 : R_1(a, b)$, $r_2 : R_1(b, c)$, $r_3 : R_1(a, c)$
$r_4 : R_1(c, d)$

$Pv(R_2(a, b)) : m_1(r_1)$
$Pv(R_2(a, c)) : m_1(r_3) + m_2(r_2 \cdot m_1(r_1))$

Provenance

Need to track from where tuples are derived, and how

Provenance Token

- base tuple: tuple id
- derived tuple: polynomial
  - binary operators $+, \cdot$
  - unary function for each mapping

Example (Provenance Tokens)

Relations R_1, R_2, Mappings
$m_1: R_1(A, B) \rightarrow R_2(A, B)$,
$m_2: R_2(A, B) \wedge R_1(B, C) \rightarrow R_2(A, C)$

$r_1: R_1(a, b)$, $r_2: R_1(b, c)$, $r_3: R_1(a, c)$
$r_4: R_1(c, d)$

$Pv(R_2(a, b)): m_1(r_1)$
$Pv(R_2(a, c)): m_1(r_3) + m_2(r_2 \cdot m_1(r_1))$
$Pv(R_2(a, d)):$
$m_2(r_4 \cdot (m_1(r_3) + m_2(r_2 \cdot m_1(r_1)))$

Provenance

Need to track from where tuples are derived, and how

Provenance Token

- base tuple: tuple id
- derived tuple: polynomial
  - binary operators $+, \cdot$
  - unary function for each mapping

Also possible: define provenance via provenance graph (omitted)

Example (Provenance Tokens)

Relations R_1, R_2, Mappings
$m_1 \colon R_1(A, B) \rightarrow R_2(A, B)$,
$m_2 \colon R_2(A, B) \wedge R_1(B, C) \rightarrow R_2(A, C)$

$r_1 \colon R_1(a, b)$, $r_2 \colon R_1(b, c)$, $r_3 \colon R_1(a, c)$
$r_4 \colon R_1(c, d)$

$Pv(R_2(a, b)) \colon m_1(r_1)$
$Pv(R_2(a, c)) \colon m_1(r_3) + m_2(r_2 \cdot m_1(r_1))$
$Pv(R_2(a, d)) \colon$
$m_2(r_4 \cdot (m_1(r_3) + m_2(r_2 \cdot m_1(r_1))))$

Infinitely many or arbitrarily large derivations $\Rightarrow$ finitely representable

Trust

Trust annotations **T** and **D** – Reject **D**

Trust

Trust annotations **T** and **D** – Reject **D**

Trust Conditions

- Define trust conditions ρ_i for mappings m_i
  - e.g. trivial conditions **T**, **D**
  - more elaborate conditions like **T** if $x_i > 4$, **D** otherwise
- Assume every base tuple to be annotated with **T**, **D**
- Import data if ρ_i is satisfied and tuples are trusted

Trust

Trust annotations **T** and **D** – Reject **D**

Trust Conditions

- Define trust conditions ρ_i for mappings m_i
  - e.g. trivial conditions **T**, **D**
  - more elaborate conditions like **T** if $x_i > 4$, **D** otherwise
- Assume every base tuple to be annotated with **T**, **D**
- Import data if ρ_i is satisfied and tuples are trusted

Evaluate (finite) provenance expressions

- Identify **T**, **D** with boolean *true*, *false*, and $+$, $\cdot$ with $\vee$, $\wedge$
- Combine trust conditions on mappings by $\wedge$ with arguments

$\Rightarrow$ Consider finite provenance expression as boolean equation

Trust

Trust annotations **T** and **D** – Reject **D**

Trust Conditions

- Define trust conditions ρ_i for mappings m_i
  - e.g. trivial conditions **T**, **D**
  - more elaborate conditions like **T** if $x_i > 4$, **D** otherwise
- Assume every base tuple to be annotated with **T**, **D**
- Import data if ρ_i is satisfied and tuples are trusted

Evaluate (finite) provenance expressions

- Identify **T**, **D** with boolean *true*, *false*, and $+$, $\cdot$ with $\vee$, $\wedge$
- Combine trust conditions on mappings by $\wedge$ with arguments

$\Rightarrow$ Consider finite provenance expression as boolean equation

Encode trust in Σ'

- add table R^t; change intern mappings to
  - $R^t(\vec{x}) = \mathit{trusted}(R^i(\vec{x}))$
  - $R^t(\vec{x}) \wedge \neg R^r(\vec{x}) \rightarrow R^o(\vec{x})$

Outline

1. Motivation
2. Query Answering in Peer Data Management
3. Materialization of Data in Peer Data Management
4. Optimization of Query Reformulation
5. Conclusion

Query Reformulation in Peer Data Integration

consider again query answering for $\mathcal{PPL}$

Query Reformulation Algorithm

- combination of LAV and GAV mappings
- for a query goal
  - unfolding if part of a GAV mapping
  - rewriting if part of a LAV mapping
- follow semantic paths through the system
- create (special) rule-goal tree

Query Reformulation in Peer Data Integration

consider again query answering for $\mathcal{PPL}$

Query Reformulation Algorithm

- combination of LAV and GAV mappings
- for a query goal
  - unfolding if part of a GAV mapping
  - rewriting if part of a LAV mapping
- follow semantic paths through the system
- create (special) rule-goal tree

$\Rightarrow$ **prune the search tree**

Query Reformulation in Peer Data Integration

consider again query answering for $\mathcal{PPL}$

Query Reformulation Algorithm

- combination of LAV and GAV mappings
- for a query goal
  - unfolding if part of a GAV mapping
  - rewriting if part of a LAV mapping
- follow semantic paths through the system
- create (special) rule-goal tree

$\Rightarrow$ **prune the search tree**

- peers described by XML schemas
- mappings described as queries in a subset of XQuery

Pruning the Search Tree

possibilities for optimization

- Pruning reformulation goals
  - identify dead ends, redundancies
- Minimizing reformulations
  - identify redundant subexpressions
- Pre-computation of semantic paths
  - a priori optimization
- Order of expansions (search strategy)
- Memorization
- Find first reformulations quickly

Pruning the Search Tree

possibilities for optimization

- Pruning reformulation goals ⇒ XML query containment
  - identify dead ends, redundancies
- Minimizing reformulations
  - identify redundant subexpressions
- Pre-computation of semantic paths
  - a priori optimization
- Order of expansions (search strategy)
- Memorization
- Find first reformulations quickly

Pruning the Search Tree

possibilities for optimization

- Pruning reformulation goals ⇒ XML query containment
  - identify dead ends, redundancies
- Minimizing reformulations ⇒ minimization of XML queries
  - identify redundant subexpressions
- Pre-computation of semantic paths
  - a priori optimization
- Order of expansions (search strategy)
- Memorization
- Find first reformulations quickly

Pruning the Search Tree

possibilities for optimization

- Pruning reformulation goals ⇒ XML query containment
  - identify dead ends, redundancies
- Minimizing reformulations ⇒ minimization of XML queries
  - identify redundant subexpressions
- Pre-computation of semantic paths ⇒ mapping composition
  - a priori optimization
- Order of expansions (search strategy)
- Memorization
- Find first reformulations quickly

Outline

Conclusion

Theory of Peer Data Management

considering PDM: interesting questions and results

- Summary
  - Peer Data Integration
    - global FO theory or "modular" semantics
  - Data Exchange in Peer Data Management
    - exchange certain answers
    - AXML (service invocations, rewritings, query answering)
    - update exchange (including trust, provenance)

- Further Results
  - Trust, Priorities, Preferences
  - (In)consistency handling
  - Updates
  - ...

References I

S. Abiteboul, O. Benjelloun, and T. Milo.
Positive active xml.
In *PODS*, pages 35–45, 2004.

D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Inconsistency tolerance in p2p data integration: An epistemic logic approach.
Inf. Syst., 33(4-5):360–384, 2008.

D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati.
Logical foundations of peer-to-peer data integration.
In *PODS*, pages 241–251, 2004.

G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
On reconciling data exchange, data integration, and peer data management.
In *PODS*, pages 133–142, 2007.

T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen.
Update exchange with mappings and provenance.
In *VLDB*, pages 675–686, 2007.

T. J. Green, G. Karvounarakis, and V. Tannen.
Provenance semirings.
In *PODS*, pages 31–40, 2007.

References II

A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov.
Schema mediation for large-scale semantic data sharing.
VLDB J., 14(1):68–83, 2005.

I. Tatarinov and A. Y. Halevy.
Efficient query reformulation in peer-data management systems.
In *SIGMOD Conference*, pages 539–550. ACM, 2004.

Thank you!