UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Özgür L. Özçep

Data Exchange 2

Lecture 7: Query Answering by Rewriting, Mapping Management
28 May 2020

Informationssysteme CS4130
(Summer 2020)

Query Answering

Remember: Certain Answers

- Given mapping $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$
- Semantics of query answering specified as certain answer semantics

Definition

The certain answers of query Q over τ for given instance $\mathfrak{S}$ is defined as

$$cert_{\mathcal{M}}(Q, \mathfrak{S}) = \bigcap \{ \, cert(Q, \mathfrak{T}) \mid \mathfrak{T} \in SOL_{\mathcal{M}}(\mathfrak{S}) \, \}$$

Remember: Certain Answers

- Given mapping $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$
- Semantics of query answering specified as certain answer semantics

Definition

The certain answers of query Q over τ for given instance $\mathfrak{S}$ is defined as

$$cert_{\mathcal{M}}(Q, \mathfrak{S}) = \bigcap \{ \, cert(Q, \mathfrak{T}) \mid \mathfrak{T} \in SOL_{\mathcal{M}}(\mathfrak{S}) \, \}$$

- We saw: In many cases it is not necessary to compute all solutions to get certain answers $\Longrightarrow$ universal solutions

Remember: Certain Answers

- Given mapping $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$
- Semantics of query answering specified as certain answer semantics

Definition

The certain answers of query Q over τ for given instance $\mathfrak{S}$ is defined as

$$cert_{\mathcal{M}}(Q, \mathfrak{S}) = \bigcap \{ \, cert(Q, \mathfrak{T}) \mid \mathfrak{T} \in SOL_{\mathcal{M}}(\mathfrak{S}) \, \}$$

- We saw: In many cases it is not necessary to compute all solutions to get certain answers $\Longrightarrow$ universal solutions
- But as universal solution $\mathfrak{T}$ (usually) is an incomplete DB, we would have to consider all completions (requires: $cert(Q, \mathfrak{T})$)

Remember: Certain Answers

- Given mapping $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$
- Semantics of query answering specified as certain answer semantics

Definition

The certain answers of query Q over τ for given instance $\mathfrak{S}$ is defined as

$$cert_{\mathcal{M}}(Q, \mathfrak{S}) = \bigcap \{ \, cert(Q, \mathfrak{T}) \mid \mathfrak{T} \in SOL_{\mathcal{M}}(\mathfrak{S}) \, \}$$

- We saw: In many cases it is not necessary to compute all solutions to get certain answers $\Longrightarrow$ universal solutions
- But as universal solution $\mathfrak{T}$ (usually) is an incomplete DB, we would have to consider all completions (requires: $cert(Q, \mathfrak{T})$)
- Sometimes this is not required $\Longrightarrow$ Query rewriting

Certain Answers Naively

Definition (Naive evaluation strategy for general DBs)

For an arbitrary general DB $\mathfrak{S}$ the set of answers following a naive evaluation strategy, for short $Q_{naive}(\mathfrak{S})$, is calculated as follows:

- Treat marked NULLS in $\mathfrak{S}$ as constants (i.e. $\perp = \perp$ is true but not $\perp = c$ and not $\perp = \perp'$)
- Calculate $Q(\mathfrak{S})$ under this perspective (treating $\mathfrak{S}$ as ordinary complete DB)
- and then eliminate all tuples from $Q(\mathfrak{S})$ containing a NULL

Certain Answers Naively

Theorem

For UCQs Q:

$$cert(\mathfrak{S}, Q) = Q_{naive}(\mathfrak{S})$$

Proof sketch:

- For every $\mathfrak{S}' \in Rep(\mathfrak{S})$ there is $\mathfrak{S} \xrightarrow{hom} \mathfrak{S}'$
- As homomorphisms preserve answers of CQs:
  $Q_{naive}(\mathfrak{S}) =$ NULL-free tuples in $Q(\mathfrak{S}) \subseteq \bigcap_{\mathfrak{S}' \in Rep(\mathfrak{S})} Q(\mathfrak{S}')$
- $Q_{naive}(\mathfrak{S}) \supseteq \bigcap_{\mathfrak{S}' \in Rep(\mathfrak{S})} Q(\mathfrak{S}')$
  because $\mathfrak{S}$ can be considered as its own completion (when treating NULLs consistently as constants).

Lit: T. Imielinski and W. Lipski, Jr. Incomplete information in relational databases. J. ACM, 31(4):761–791, Sept. 1984.

Use of naive strategy for DE

Definition (Naive Evaluation Strategy for DEs)

$$cert_{\mathcal{M}}(\mathfrak{S}, Q) = Q_{naive}(\mathfrak{T})$$

where $\mathfrak{T}$ is a universal solution for $\mathcal{M}$ and $\mathfrak{S}$.

Use of naive strategy for DE

Definition (Naive Evaluation Strategy for DEs)

$$cert_{\mathcal{M}}(\mathfrak{S}, Q) = Q_{naive}(\mathfrak{T})$$

where $\mathfrak{T}$ is a universal solution for $\mathcal{M}$ and $\mathfrak{S}$.

- This strategy works also for Datalog programs as constraints for the target schema τ
  - Reason: Datalog programs are preserved under homomorphisms
  - Even if one adds inequalities, naive evaluation works
  - Hence certain answering is here in PTime

Rewritability

- Naive evaluation is a form of rewriting
- Again: Fundamental method that re-appears in different areas of CS
- Rewrite a query w.r.t. a given KB into a new query that "contains" the knowledge of KB

- Challenges
  - Preserve the semantics in the rewriting process: ensure correctness (easy) and completeness (difficult)
  - The language of the output query is constraint to a "simple language" (so rewritability not always guaranteed)

Rewritability for DE

Definition (FOL Rewritability)

Let $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$ be a mapping and Q be a query over τ.

Then Q is said to be FOL-rewritable over canonical universal solutions ($\mathfrak{T}$) under $\mathcal{M}$ iff there is a FOL query Q_{rew} over τ^C s.t.

$$cert_{\mathcal{M}}(Q, \mathfrak{S}) = Q_{rew}(\mathfrak{T})$$

- Here $\tau^C = \tau \cup \{C\}$ where
  unary predicate C depicts all constants (not NULLs) in targets
- C works like a type predicate

Rewritability for DE

Definition (FOL Rewritability)

Let $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$ be a mapping and Q be a query over τ.

Then Q is said to be FOL-rewritable over canonical universal solutions ($\mathfrak{T}$) under $\mathcal{M}$ iff there is a FOL query Q_{rew} over τ^C s.t.

$$cert_{\mathcal{M}}(Q, \mathfrak{S}) = Q_{rew}(\mathfrak{T})$$

Note: One must find one rewriting for any given pair of source $\mathfrak{S}$ and universal solution $\mathfrak{T}$

- The known component is the mapping $\mathcal{M}$
- The unknown components are all pairs $(\mathfrak{S}, \mathfrak{T})$

Rewritability for DE

Definition (FOL Rewritability)

Let $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$ be a mapping and Q be a query over τ.

Then Q is said to be FOL-rewritable over canonical universal solutions under $\mathcal{M}$ iff there is a FOL query Q_{rew} over τ^C such that

$$cert_{\mathcal{M}}(Q, \mathfrak{S}) = Q_{rew}(\mathfrak{T})$$

If, in the definition, one talks about cores $\mathfrak{T}$ instead of universal solutions then Q is said to be FOL-rewritable over cores

Theorem

For mappings without target dependencies:
FOL-rewrit. over core $\models$ *FOL-rewrit. over universal solution,*
but not vice versa.

Rewritability for DE

Definition (FOL-Rewritability)

Let $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$ be a mapping and Q be a query over τ.

Then Q is said to be FOL-rewritable over canonical universal solutions under $\mathcal{M}$ iff there is a FOL query Q_{rew} over τ^C such that

$$cert_{\mathcal{M}}(Q, \mathfrak{S}) = Q_{rew}(\mathfrak{T})$$

Example

- $Q(\vec{x})$: a conjunctive query
- Q_{rew}: $Q(\vec{x}) \wedge C(x_1) \wedge \cdots \wedge C(x_n)$
  This is actually the syntactic form of Q_{naive}
- The rewriting is even independent of $\mathcal{M}$
- So: (U)CQs are rewritable for any mapping

Adding Negations to Query Language

- Negations in query languages lead to loss of naive rewriting technique
- Even if one allows negation only within inequalities

Definition (Conjunctive Queries with inequalities $CQ^{\neq}$)

A conjunctive query with inequalities is a query of the form

$$Q(\vec{x}) = \exists \vec{y} \left(\alpha_1(\vec{x_1}, \vec{y_1}) \wedge \cdots \wedge \alpha_n(\vec{x_n}, \vec{y_n})\right)$$

where α_i is either an atomic relational formula or an inequality $z_i \neq z_j$.

Example (No Naive Evaluation Possible)

Source DB

Flight (src, dest, airl, dep)

src	dest	airl	dep
paris	sant.	airFr	2320
paris	sant.	lan	2200

Target DB

Routes(<u>fno</u>, src, dest)

Info(<u>fno</u>, dep, arr, airl)

- **Dependencies** $M_{\sigma\tau}$

$$\begin{aligned}&Flight(src, dest, airl, dep) \longrightarrow \\ &\quad \exists fno \, \exists \, arr(Routes(fno, src, dest) \wedge Info(fno, dep, arr, airl))\end{aligned}$$

Example (No Naive Evaluation Possible)

Source DB

Flight (src, dest, airl, dep)

src	dest	airl	dep
paris	sant.	airFr	2320
paris	sant.	lan	2200

Target DB

Routes(<u>fno</u>, src, dest)

Info(<u>fno</u>, dep, arr, airl)

- **Dependencies** $M_{\sigma\tau}$

$$
\begin{array}{l}
Flight(src, dest, airl, dep) \longrightarrow \\
\quad \exists fno \, \exists \, arr(Routes(fno, src, dest) \wedge Info(fno, dep, arr, airl))
\end{array}
$$

- Any universal solution $\mathfrak{T}'$ contains as sub-instance universal τ-**solution**

$$
\begin{array}{rcl}
\mathfrak{T} & = & \{ \, Routes(\perp_1, paris, sant), \; Info(\perp_1, 2320, \perp_2, airFr), \\
 & & \;\; Routes(\perp_3, paris, sant), \; Info(\perp_3, 2320, \perp_4, lan) \, \}
\end{array}
$$

Example (No Naive Evaluation Possible)

Source DB

Flight (	src,	dest,	airl,	dep	)
	paris	sant.	airFr	2320	
	paris	sant.	lan	2200	

Target DB

Routes(<u>fno</u>, src, dest)

Info(<u>fno</u>, dep, arr, airl)

- **Dependencies** $M_{\sigma\tau}$

$$\begin{aligned}&\mathit{Flight}(\mathit{src}, \mathit{dest}, \mathit{airl}, \mathit{dep}) \longrightarrow \\ &\quad \exists \mathit{fno}\ \exists\ \mathit{arr}(\mathit{Routes}(\mathit{fno}, \mathit{src}, \mathit{dest}) \wedge \mathit{Info}(\mathit{fno}, \mathit{dep}, \mathit{arr}, \mathit{airl}))\end{aligned}$$

- Any universal solution $\mathfrak{T}'$ contains as sub-instance universal τ-**solution**

$$\begin{aligned}\mathfrak{T} \quad = \quad \{ &\mathit{Routes}(\bot_1, \mathit{paris}, \mathit{sant}),\ \mathit{Info}(\bot_1, 2320, \bot_2, \mathit{airFr}), \\ &\mathit{Routes}(\bot_3, \mathit{paris}, \mathit{sant}),\ \mathit{Info}(\bot_3, 2320, \bot_4, \mathit{lan}) \}\end{aligned}$$

- Query $Q(x, z) = \exists y \exists y' (\mathit{Routes}(y, x, z) \wedge \mathit{Routes}(y', x, z) \wedge y \neq y')$

Example (No Naive Evaluation Possible)

Source DB

Flight (	src,	dest,	airl,	dep	)
	paris	sant.	airFr	2320	
	paris	sant.	lan	2200	

Target DB

Routes(<u>fno</u>, src, dest)

Info(<u>fno</u>, dep, arr, airl)

- **Dependencies** $M_{\sigma\tau}$

$$Flight(src, dest, airl, dep) \longrightarrow$$
$$\exists fno\ \exists\ arr(Routes(fno, src, dest) \wedge Info(fno, dep, arr, airl))$$

- Any universal solution $\mathfrak{T}'$ contains as sub-instance universal τ-**solution**

$$\begin{aligned} \mathfrak{T} \quad = \quad \{\ & Routes(\bot_1, paris, sant),\ Info(\bot_1, 2320, \bot_2, airFr), \\ & Routes(\bot_3, paris, sant),\ Info(\bot_3, 2320, \bot_4, lan)\ \} \end{aligned}$$

- Query $Q(x, z) = \exists y \exists y'(Routes(y, x, z) \wedge Routes(y', x, z) \wedge y \neq y')$
- $Q_{naive}(\mathfrak{T}') = \{(paris, sant)\}$ (for any universal solution $\mathfrak{T}'$)

Example (No Naive Evaluation Possible)

Source DB

Flight (	src,	dest,	airl,	dep	)
	paris	sant.	airFr	2320	
	paris	sant.	lan	2200	

Target DB

Routes(<u>fno</u>, src, dest)

Info(<u>fno</u>, dep, arr, airl)

- **Dependencies** $M_{\sigma\tau}$

$$\begin{array}{l} \mathit{Flight}(\mathit{src}, \mathit{dest}, \mathit{airl}, \mathit{dep}) \longrightarrow \\ \quad \exists \mathit{fno}\ \exists\ \mathit{arr}(\mathit{Routes}(\mathit{fno}, \mathit{src}, \mathit{dest}) \wedge \mathit{Info}(\mathit{fno}, \mathit{dep}, \mathit{arr}, \mathit{airl})) \end{array}$$

- Any universal solution $\mathfrak{T}'$ contains as sub-instance universal τ-**solution**

$$\begin{array}{rcl} \mathfrak{T} & = & \{\ \mathit{Routes}(\bot_1, \mathit{paris}, \mathit{sant}),\ \mathit{Info}(\bot_1, 2320, \bot_2, \mathit{airFr}), \\ & & \quad \mathit{Routes}(\bot_3, \mathit{paris}, \mathit{sant}),\ \mathit{Info}(\bot_3, 2320, \bot_4, \mathit{lan})\ \} \end{array}$$

- Query $Q(x, z) = \exists y \exists y' (\mathit{Routes}(y, x, z) \wedge \mathit{Routes}(y', x, z) \wedge y \neq y')$
- $Q_{naive}(\mathfrak{T}') = \{(\mathit{paris}, \mathit{sant})\}$ (for any universal solution $\mathfrak{T}'$)
- But: $\mathit{cert}_{\mathcal{M}}(Q(x, z), \mathfrak{S}) = \emptyset$ because there is a solution

$$\begin{array}{rcl} \mathfrak{T}'' & = & \{\ \mathit{Routes}(\bot_1, \mathit{paris}, \mathit{sant}),\ \mathit{Info}(\bot_1, 2320, \bot_2, \mathit{airFr}), \\ & & \quad \mathit{Info}(\bot_1, 2320, \bot_2, \mathit{lan})\ \} \end{array}$$

$CQ^{\neq}$ is in coNP

- In case of $CQ^{\neq}$ one cannot even find a tractable means to answer them w.r.t. certain answer semantics

Theorem

Let $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$ be a mapping where M_τ is the union of egds and weakly acyclic tgds, and let Q be a $UCQ^{\neq}$ query. Then:

$CERTAIN_{\mathcal{M}}(Q)$ is in coNP

Non-rewritability

- Generally it is not possible to decide whether rewritability holds

Theorem

For mappings without target constraints one can not decide whether a given FOL query is rewritable over the canonical solutions (over the core).

- Showing Non-FOL-rewritability can be done with locality tools
- Actually: One uses (adapted) Hanf-locality

Not Covered in our DE Lectures

- Different semantics for query answering
  - Combinations of open-world (certain answers) and closed-word semantics
- DE for non-relational DBs
  - e.g., DE for semi-structured data (XML)
  - requires techniques other than that for relational DE

Not Covered in our DE Lectures

- Different semantics for query answering
  - Combinations of open-world (certain answers) and closed-word semantics
- DE for non-relational DBs
  - e.g., DE for semi-structured data (XML)
  - requires techniques other than that for relational DE
- Rest of this lecture: mapping management
  - How to maintain mappings w.r.t. consistency (only a few remarks today)
  - How to compose mappings
  - How to invert mappings: Get back source DB from target DB

Motivation Mapping Management

Consistency of Mappings

- So far: Considered existence of τ-solutions given σ-instance in mapping $\mathcal{M}$
- Now: Given only $\mathcal{M}$
  - consistency/local consistency of $\mathcal{M}$: Is there a σ-instance s.t. there is a τ-solution
  - Absolute consistency/Global consistency: Is there for each σ-instance a τ-solution?

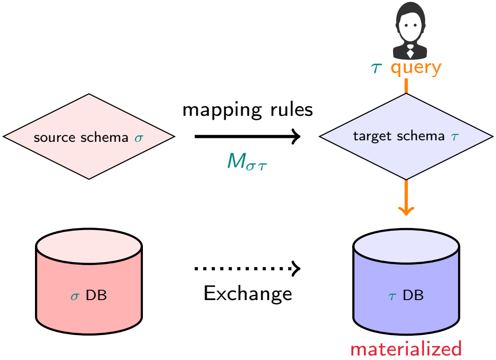
Mapping Evolution

- Mappings may change due to schema evolution
  - Target schema changes: need composition of mappings
  - Source schema changes: need inverse of mappings

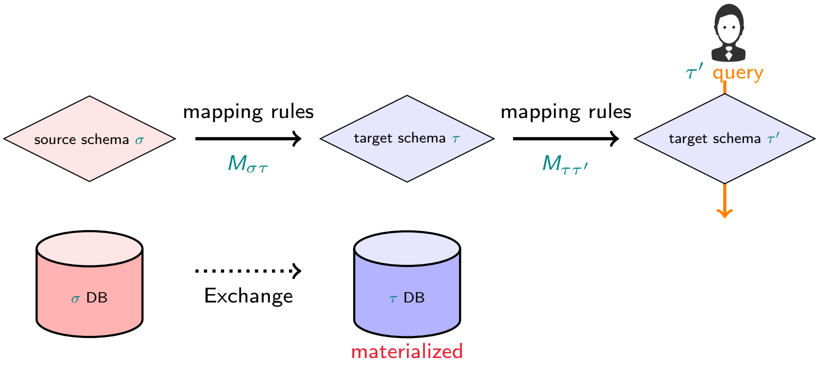
Mapping Evolution

- Mappings may change due to schema evolution
  - Target schema changes: need composition of mappings
  - Source schema changes: need inverse of mappings
  - Can think of other operations (merge of mappings . . .)

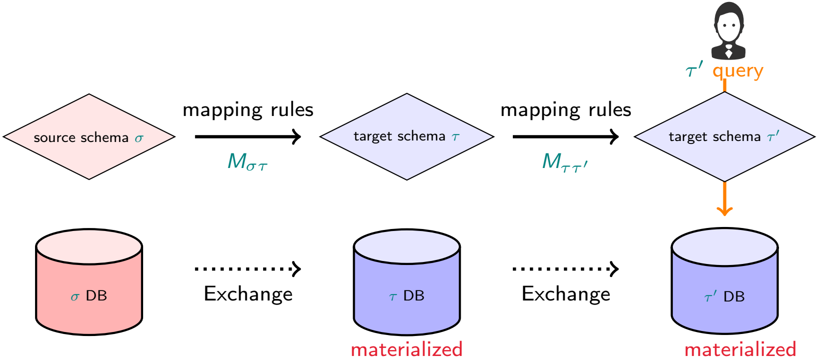
Composition for Target Schema Change



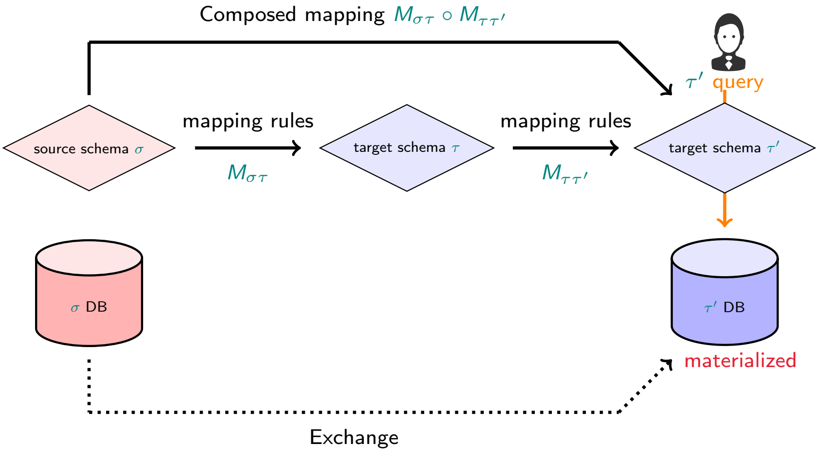
Composition for Target Schema Change



Composition for Target Schema Change



Composition for Target Schema Change



Example (DE in Flight Domain)

Source schema σ

Geo(	city,	coun,	pop)	
Flight(	src,	dest,	airl,	dep)

Target schema τ

Route(	<u>fno</u>,	src,	dest)	
Info(	<u>fno</u>,	dep,	arr,	airl)
Serves(	airl,	city,	coun,	phone)

Example (DE in Flight Domain)

Source schema σ

Geo(city, coun, pop)
Flight(src, dest, airl, dep)

Target schema τ

Route(<u>fno</u>, src, dest)
Info(<u>fno</u>, dep, arr, airl)
Serves(airl, city, coun, phone)

Mapping rules $M_{\sigma\tau}$

1. $\mathit{Flight}(\mathit{src}, \mathit{dest}, \mathit{airl}, \mathit{dep}) \longrightarrow \exists \mathit{fno}\ \exists \mathit{arr}\ (\mathit{Route}(\mathit{fno}, \mathit{src}, \mathit{dest}) \wedge \mathit{Info}(\mathit{fno}, \mathit{dep}, \mathit{arr}, \mathit{airl}))$
2. $\mathit{Flight}(\mathit{city}, \mathit{dest}, \mathit{airl}, \mathit{dep}) \wedge \mathit{Geo}(\mathit{city}, \mathit{coun}, \mathit{pop}) \longrightarrow \exists \mathit{phone}\ (\mathit{Serves}(\mathit{airl}, \mathit{city}, \mathit{coun}, \mathit{phone}))$
3. $\mathit{Flight}(\mathit{src}, \mathit{city}, \mathit{airl}, \mathit{dep}) \wedge \mathit{Geo}(\mathit{city}, \mathit{coun}, \mathit{pop}) \longrightarrow \exists \mathit{phone}\ (\mathit{Serves}(\mathit{airl}, \mathit{city}, \mathit{coun}, \mathit{phone}))$

Example (DE in Flight Domain)

Source schema σ

Geo(city, coun, pop)
Flight(src, dest, airl, dep)

Target schema τ

Route(<u>fno</u>, src, dest)
Info(<u>fno</u>, dep, arr, airl)
Serves(airl, city, coun, phone)

Mapping rules $M_{\sigma\tau}$

1. $Flight(src, dest, airl, dep) \longrightarrow \exists fno\ \exists arr\ (Route(fno, src, dest) \wedge Info(fno, dep, arr, airl))$
2. $Flight(city, dest, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow \exists phone\ (Serves(airl, city, coun, phone))$
3. $Flight(src, city, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow \exists phone\ (Serves(airl, city, coun, phone))$

New target schema τ'

InfoAirline(airline, city, coun, phone, year)
InfoJourney(<u>fno</u>, source, dep, dest, arr, airl)

Example (DE in Flight Domain)

Source schema σ

Geo(	city,	coun,	pop)	
Flight(	src,	dest,	airl,	dep)

Target schema τ

Route(	<u>fno</u>,	src,	dest)	
Info(	<u>fno</u>,	dep,	arr,	airl)
Serves(	airl,	city,	coun,	phone)

Mapping rules $M_{\sigma\tau}$

1. $Flight(src, dest, airl, dep) \longrightarrow \exists fno\ \exists arr\ (Route(fno, src, dest) \wedge Info(fno, dep, arr, airl))$
2. $Flight(city, dest, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow \exists phone\ (Serves(airl, city, coun, phone))$
3. $Flight(src, city, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow \exists phone\ (Serves(airl, city, coun, phone))$

New target schema τ'

InfoAirline(	airline,	city,	coun,	phone,	year)	
InfoJourney(	<u>fno</u>,	source,	dep,	dest,	arr,	airl)

Mapping rules $M_{\tau\tau'}$

1. $Serves(airl, city, coun, phone) \longrightarrow \exists year\ InfoAirline(airl, city, coun, phone, year)$
2. $Route(fno, src, dest) \wedge Info(fno, dep, arr, airl) \longrightarrow InfoJourney(fno, dep, dest, arr, airl)$

Example (DE in Flight Domain)

Source schema σ

Geo(city, coun, pop)
Flight(src, dest, airl, dep)

Target schema τ

Route(<u>fno</u>, src, dest)
Info(<u>fno</u>, dep, arr, airl)
Serves(airl, city, coun, phone)

Mapping rules $M_{\sigma\tau}$

1. $Flight(src, dest, airl, dep) \longrightarrow \exists fno\ \exists arr\ (Route(fno, src, dest) \wedge Info(fno, dep, arr, airl))$
2. $Flight(city, dest, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow \exists phone\ (Serves(airl, city, coun, phone))$
3. $Flight(src, city, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow \exists phone\ (Serves(airl, city, coun, phone))$

New target schema τ'

InfoAirline(airline, city, coun, phone, year)
InfoJourney(<u>fno</u>, source, dep, dest, arr, airl)

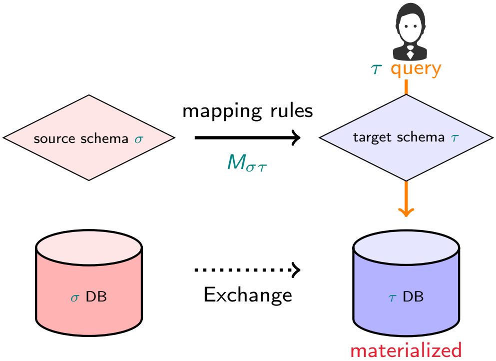
Mapping rules $M_{\tau\tau'}$

1. $Serves(airl, city, coun, phone) \longrightarrow \exists year\ InfoAirline(airl, city, coun, phone, year)$
2. $Route(fno, src, dest) \wedge Info(fno, dep, arr, airl) \longrightarrow InfoJourney(fno, dep, dest, arr, airl)$

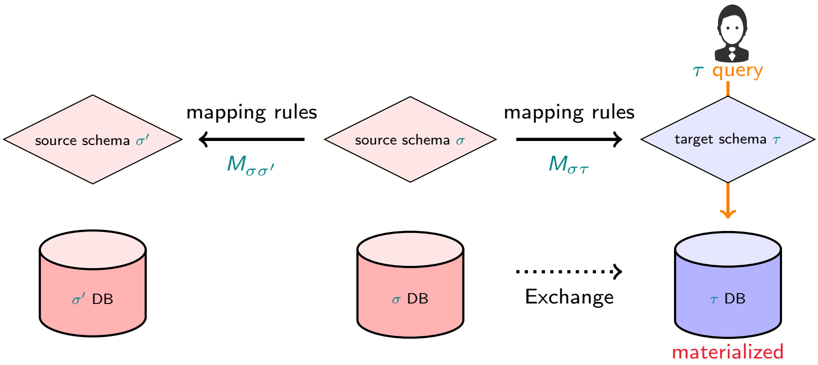
Composed rules $M_{\sigma\tau} \circ M_{\tau\tau'}$

1. $Flight(src, dest, airl, dep) \longrightarrow \exists fno\ \exists arr\ (InfoJourney(fno, src, dep, dest, arr, airl))$
2. $Flight(city, dest, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow$
   $\exists phone\ \exists year\ InfoAirline(airl, city, coun, phone, year)$
3. $Flight(src, city, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow$
   $\exists phone\ \exists year\ InfoAirline(airl, city, coun, phone, year)$

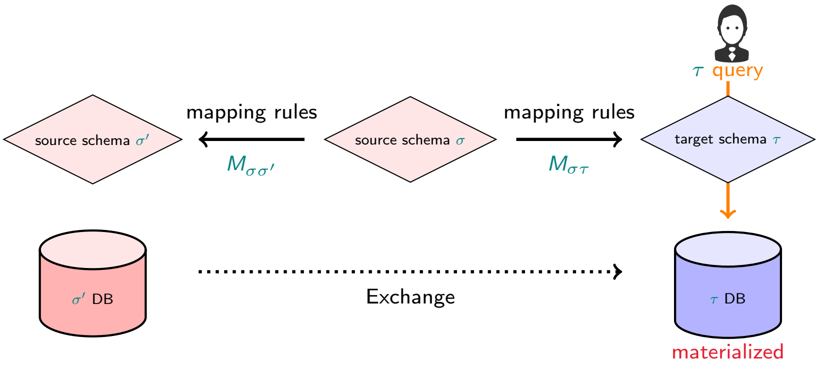
Inverse for Source Schema Change



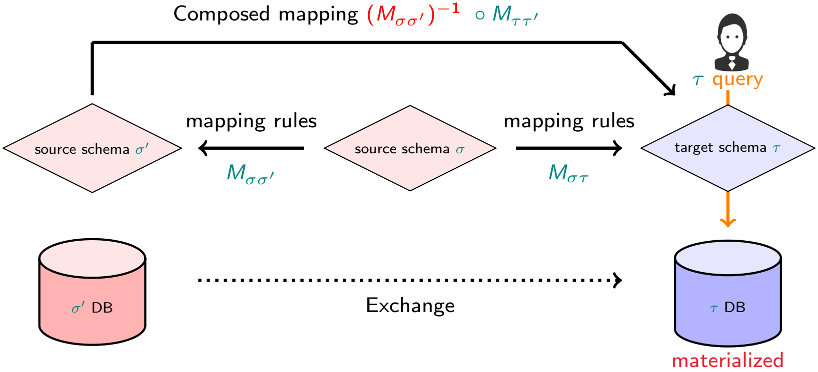
Inverse for Source Schema Change



Inverse for Source Schema Change



Inverse for Source Schema Change



Main question: Closure

- Are mappings closed under
  - composition?
  - inverse?
- In general they are not
- Solution: Use second order logic with Skolem functions

Mapping Composition

- Treat mappings as binary relations
  $[\![\mathcal{M}_{\tau_1\tau_2}]\!]$ = set of pairs (source τ_1-instance, τ_2-solution)

- Treat mappings as binary relations
  $[\![\mathcal{M}_{\tau_1\tau_2}]\!]$ = set of pairs (source τ_1-instance, τ_2-solution)

Definition (Mapping composition)

Given schemata σ, τ, τ' and mappings $\mathcal{M}_{\sigma\tau}$, $\mathcal{M}_{\tau\tau'}$. The composition of $\mathcal{M}_{\sigma\tau}, \mathcal{M}_{\tau\tau'}$ is defined by

$$\begin{aligned}[\![\mathcal{M}_{\sigma\tau}]\!] \circ [\![\mathcal{M}_{\tau\tau'}]\!] \;=\; \{(\mathfrak{S}, \mathfrak{T}') \mid & \text{ there is } \tau\text{-instance } \mathfrak{T} \text{ s.t.} \\ & (\mathfrak{S}, \mathfrak{T}) \in [\![\mathcal{M}_{\sigma\tau}]\!] \text{ and } (\mathfrak{T}, \mathfrak{T}') \in [\![\mathcal{M}_{\tau\tau'}]\!]\}\end{aligned}$$

- Treat mappings as binary relations
  $[\![\mathcal{M}_{\tau_1\tau_2}]\!]$ = set of pairs (source τ_1-instance, τ_2-solution)

Definition (Mapping composition)

Given schemata σ, τ, τ' and mappings $\mathcal{M}_{\sigma\tau}$, $\mathcal{M}_{\tau\tau'}$. The composition of $\mathcal{M}_{\sigma\tau}, \mathcal{M}_{\tau\tau'}$ is defined by

$$\begin{aligned} [\![\mathcal{M}_{\sigma\tau}]\!] \circ [\![\mathcal{M}_{\tau\tau'}]\!] \quad = \quad & \{(\mathfrak{S}, \mathfrak{T}') \mid \text{ there is } \tau\text{-instance } \mathfrak{T} \text{ s.t.} \\ & (\mathfrak{S}, \mathfrak{T}) \in [\![\mathcal{M}_{\sigma\tau}]\!] \text{ and } (\mathfrak{T}, \mathfrak{T}') \in [\![\mathcal{M}_{\tau\tau'}]\!]\} \end{aligned}$$

- Note: Semantics of composition does not say whether there exist rule set M representing $[\![\mathcal{M}_{\sigma\tau}]\!] \circ [\![\mathcal{M}_{\tau\tau'}]\!]$.
  (That is the whole point of the closure problem)

Example

- $\sigma : \{Takes(name, course)\}$
- $\tau : \{Takes1(name, course), Student(name, sid)\}$
- $\tau' : \{Enrolled(sid, course)\}$
- $\mathcal{M}_{\sigma\tau}$:
  $\{Takes(n, c) \to Takes1(n, c), Takes(n, c) \to \exists s Student(n, s)\}$
- $\mathcal{M}_{\tau\tau'} : \{Student(n, s) \wedge Takes1(n, c) \to Enrolled(s, c)\}$

Example

- $\sigma : \{Takes(name, course)\}$
- $\tau : \{Takes1(name, course), Student(name, sid)\}$
- $\tau' : \{Enrolled(sid, course)\}$
- $\mathcal{M}_{\sigma\tau}$:
  $\{Takes(n, c) \rightarrow Takes1(n, c), Takes(n, c) \rightarrow \exists s Student(n, s)\}$
- $\mathcal{M}_{\tau\tau'} : \{Student(n, s) \wedge Takes1(n, c) \rightarrow Enrolled(s, c)\}$

- No st-tgd represents $\mathcal{M}_{\sigma\tau} \circ \mathcal{M}_{\tau\tau'}$, in particular not st-tgd:

$$Takes(n, c) \rightarrow \exists y Enrolled(y, c)$$

Example

- $\sigma : \{Takes(name, course)\}$
- $\tau : \{Takes1(name, course), Student(name, sid)\}$
- $\tau' : \{Enrolled(sid, course)\}$
- $\mathcal{M}_{\sigma\tau}$:
  $\{Takes(n, c) \rightarrow Takes1(n, c), Takes(n, c) \rightarrow \exists s Student(n, s)\}$
- $\mathcal{M}_{\tau\tau'} : \{Student(n, s) \wedge Takes1(n, c) \rightarrow Enrolled(s, c)\}$

- No st-tgd represents $\mathcal{M}_{\sigma\tau} \circ \mathcal{M}_{\tau\tau'}$, in particular not st-tgd:

$$Takes(n, c) \rightarrow \exists y Enrolled(y, c)$$

- Intuitively need to express dependency $f : n \rightarrow sid$

$$Takes(n, c) \rightarrow Enrolled(f(n), c)$$

Example

- $\sigma : \{Takes(name, course)\}$
- $\tau : \{Takes1(name, course), Student(name, sid)\}$
- $\tau' : \{Enrolled(sid, course)\}$
- $\mathcal{M}_{\sigma\tau}$:
  $\{Takes(n, c) \rightarrow Takes1(n, c), Takes(n, c) \rightarrow \exists s Student(n, s)\}$
- $\mathcal{M}_{\tau\tau'} : \{Student(n, s) \wedge Takes1(n, c) \rightarrow Enrolled(s, c)\}$

- No st-tgd represents $\mathcal{M}_{\sigma\tau} \circ \mathcal{M}_{\tau\tau'}$, in particular not st-tgd:

$$Takes(n, c) \rightarrow \exists y Enrolled(y, c)$$

- Intuitively need to express dependency $f : n \rightarrow sid$

$$Takes(n, c) \rightarrow Enrolled(f(n), c)$$

- f called Skolem function

Complexity of Relational Composition

Problem $\mathit{COMPOSITION}(\mathcal{M}_{\sigma\tau}, \mathcal{M}_{\tau\tau'})$

- INPUT: Instance $\mathfrak{S}$ of σ and instance $\mathfrak{T}'$ of τ'
- Output: Is $(\mathfrak{S}, \mathfrak{T}') \in [\![\mathcal{M}_{\sigma\tau}]\!] \circ [\![\mathcal{M}_{\tau\tau'}]\!]$?

Complexity of Relational Composition

Problem *COMPOSITION*$(\mathcal{M}_{\sigma\tau}, \mathcal{M}_{\tau\tau'})$

- INPUT: Instance $\mathfrak{S}$ of σ and instance $\mathfrak{T}'$ of τ'
- Output: Is $(\mathfrak{S}, \mathfrak{T}') \in [\![\mathcal{M}_{\sigma\tau}]\!] \circ [\![\mathcal{M}_{\tau\tau'}]\!]$?

Theorem

- *For mappings $\mathcal{M}_{\sigma\tau}$ and $\mathcal{M}_{\tau\tau'}$ specified by st-tgds, COMPOSITION$(\mathcal{M}_{\sigma\tau}, \mathcal{M}_{\tau\tau'})$ is NP.*
- *One can find $\mathcal{M}^*_{\sigma\tau}$ and $\mathcal{M}^*_{\tau\tau'}$ represented by st-tgds for which COMPOSITION$(\mathcal{M}^*_{\sigma\tau}, \mathcal{M}^*_{\tau\tau'})$ is NP-complete.*

Proof by reducing from NP-hard problem of 3-colorability

Non-closure of FOL

Corollary

For the mappings $\mathcal{M}^*_{\sigma\tau}$ and $\mathcal{M}^*_{\tau\tau'}$ specified by st-tgds there is no finite set of FOL formulae representing their composition.

Proof sketch

- Assume for contradiction there is set X of FOL formulae for the composition.
- Then the NP-hard $COMPOSITION(\mathcal{M}^*_{\sigma\tau}, \mathcal{M}^*_{\tau\tau'})$ reduces to checking $(\mathfrak{S}, \mathfrak{T}') \models X$
- which is in AC^0
- But $AC^0 \subsetneq NP$, $\lightning$.

Definition (SO tgds)

Given disjoint schemata σ, τ, a second-order tuple-generating dependency from σ to τ is a formula of the form

$$\exists f_1 \ldots \exists f_m \big(\forall \vec{x}_1(\phi_1 \rightarrow \psi_1) \wedge \cdots \wedge \forall \vec{x}_n(\phi_n \rightarrow \psi_n)\big)$$

where

- each f_i is a function symbol
- each ϕ_i is conjunction of relational formulae $R(y_1, \ldots, y_k)$ or identities $t = t'$ with y_j from $\vec{x}$ and t, t' are terms built from $\{\vec{x}_i, f_1, \ldots, f_m\}$
- ψ_i is conjunction of form $R(t_1, \ldots, t_l)$ and t_j built from $\{\vec{x}_i, f_1, \ldots, f_m\}$
- each variable in $\vec{x}_i$ appears in some relational atom of ϕ_i

$f_1, \ldots, f_m$ are called Skolem functions

Semantics of SO tgds

- As in second order logic but requiring that (k-ary) fs are interpreted by k-ary functions of form

$$f : (CONST \cup VAR)^k \longrightarrow CONST \cup VAR$$

SO tgds do the job

Theorem

- *For mappings* $\mathcal{M}_{\sigma\tau}$ *and* $\mathcal{M}_{\tau\tau'}$ *specified by SO tgds* $\Sigma_{\sigma\tau}$, $\Sigma_{\tau\tau'}$, *resp., there is a set of SO tgds representing* $[\![\mathcal{M}_{\sigma\tau}]\!] \circ [\![\mathcal{M}_{\tau\tau'}]\!]$.
- *Moreover there is an exponential-time algorithm computing the composition.*

SO tgds do the job

Theorem

- *For mappings $\mathcal{M}_{\sigma\tau}$ and $\mathcal{M}_{\tau\tau'}$ specified by SO tgds $\Sigma_{\sigma\tau}$, $\Sigma_{\tau\tau'}$, resp., there is a set of SO tgds representing $[\![\mathcal{M}_{\sigma\tau}]\!] \circ [\![\mathcal{M}_{\tau\tau'}]\!]$.*
- *Moreover there is an exponential-time algorithm computing the composition.*

- This theorem applicable to mappings described by FOL st-tgds: Transform st-tgds into SO tgds using skolemization

Composing relational schema mappings

Require: on the source side reuse of variables only in equalities
Input : $\Sigma_{\sigma\tau}$, $\Sigma_{\tau\tau'}$
Output : $\Sigma_{\sigma\tau'}$

$\Sigma_{\sigma\tau'} := \emptyset$;
$m := max_{\phi \rightarrow \psi \in \Sigma_{\tau\tau'}} ||\phi||$;
forall $\underline{\phi_1 \rightarrow \pi_1, \ldots, \phi_k \rightarrow \pi_k \in \Sigma_{\sigma\tau},\ k \leq m}$ **do**
  in case of repetitions rename variables;
  $\rho := \pi_1 \wedge \cdots \wedge \pi_k$;
  forall $\underline{\pi \wedge \alpha \rightarrow \pi' \in \Sigma_{\tau\tau'}}$ and all homomorphisms $h : \pi \rightarrow \rho$ **do**
    $\Sigma_{\sigma\tau'} = \Sigma_{\sigma\tau'} \cup \{\phi_1 \wedge \cdots \wedge \phi_k \wedge h(\alpha) \rightarrow h(\pi')\}$
  end
end
return $\underline{\Sigma_{\sigma\tau'}}$;

Notation used in algorithm

- $||\phi||$ = number of atoms in ϕ
- use π for conjunctions of relational atoms and α for equality atoms
- So each SO tgd can be written as $\pi \wedge \alpha \rightarrow \pi'$

Inverting Mappings

First Definition of Inverse

- Harder than composition.
- Intuition: $\mathcal{M} \circ \mathcal{M}^{-1} =$ "identity mapping" *ID*
- But even semantics not clear: what should *ID* be?
- Let us start with

Definition (Inverse)

The mapping $\mathcal{M}_{\tau\sigma}^{-1}$ is an inverse of mapping $\mathcal{M}_{\sigma\tau}$ iff

$$\mathcal{M}_{\sigma\tau} \circ \mathcal{M}_{\tau\sigma}^{-1} = \{(\mathfrak{S}, \mathfrak{S}') \mid \mathfrak{S}, \mathfrak{S}' \text{ are } \sigma\text{-instances with } \mathfrak{S} \subseteq \mathfrak{S}'\}$$

Example

- Inverses may not be unique
  - $\mathcal{M}_{\sigma\tau} : S(x) \rightarrow T(x), S(x) \rightarrow T'(x)$
  - First inverse $\mathcal{M}_{\tau\sigma}^{-1} : T(x) \rightarrow S(x)$.
  - Another inverse: $\mathcal{M}_{\tau\sigma}^{-1} : T'(x) \rightarrow S(x)$.

- Inverse of union requires disjunction
  - $\mathcal{M}_{\sigma\tau} : S(x) \rightarrow T(x), S'(x) \rightarrow T(x)$
  - $\mathcal{M}_{\tau\sigma}^{-1} : T(x) \rightarrow S(x) \vee S'(x)$
  - So inverse (in some mapping language such as st-tgd) may not exist

    $\Longrightarrow$ Criteria for existence of inverse mappings

Subset property

Definition (Subset property)

Mapping $\mathcal{M}_{\sigma\tau}$ satisfies the subset property iff for all pairs $(\mathfrak{S}, \mathfrak{S}')$:

$$\text{If } \mathit{Sol}_{\mathcal{M}_{\sigma\tau}}(\mathfrak{S}) \subseteq \mathit{Sol}_{\mathcal{M}_{\sigma\tau}}(\mathfrak{S}') \text{ then } \mathfrak{S}' \subseteq \mathfrak{S}$$

Theorem

Let $\mathcal{M}_{\sigma\tau}$ be specified by a set of st-tgds. Then it is invertible iff it fulfils the subset property.

Complexity of Checking Invertibility

Theorem

Let $\mathcal{M}_{\sigma\tau}$ be specified by a set of st-tgds. Checking invertibility is coNP-complete.

Surprisingly the seemingly simpler problem is not decidable:

Theorem

Let $\mathcal{M}_{\sigma\tau}$ and $\mathcal{M}'_{\tau\sigma}$ be specified by finite sets of st-tgds. It is undecidable whether $\mathcal{M}'_{\tau\sigma}$ is an inverse of $\mathcal{M}_{\sigma\tau}$

Relaxed Notions of Invertibility

- Quasi-inverse
  - Not considered here, because
  - even for this relaxed notion existence of st-tgd mappings not guaranteed
- We consider notion of (maximum) recover
  - Recover sound information w.r.t. mappings
  - Existence of covers guaranteed

Definition (Recovery)

A mapping $\mathcal{M}' = \mathcal{M}'_{\tau\sigma}$ is a

- recovery of mapping $\mathcal{M} = \mathcal{M}_{\sigma\tau}$ iff for every σ instance $\mathfrak{S}$ on which $\mathcal{M}$ is defined (for short: $\mathfrak{S} \in \mathit{Dom}(\mathcal{M})$) it holds that $(\mathfrak{S}, \mathfrak{S}) \in \mathcal{M} \circ \mathcal{M}'$.
- maximum recovery of mapping $\mathcal{M}_{\sigma\tau}$ iff it is a recovery and is maximal: for every recovery $\mathcal{M}''$ of $\mathcal{M}$ it holds that $\mathcal{M} \circ \mathcal{M}' \subseteq \mathcal{M} \circ \mathcal{M}''$

Definition (Recovery)

A mapping $\mathcal{M}' = \mathcal{M}'_{\tau\sigma}$ is a

- recovery of mapping $\mathcal{M} = \mathcal{M}_{\sigma\tau}$ iff for every σ instance $\mathfrak{S}$ on which $\mathcal{M}$ is defined (for short: $\mathfrak{S} \in Dom(\mathcal{M})$) it holds that $(\mathfrak{S}, \mathfrak{S}) \in \mathcal{M} \circ \mathcal{M}'$.
- maximum recovery of mapping $\mathcal{M}_{\sigma\tau}$ iff it is a recovery and is maximal: for every recovery $\mathcal{M}''$ of $\mathcal{M}$ it holds that $\mathcal{M} \circ \mathcal{M}' \subseteq \mathcal{M} \circ \mathcal{M}''$

- The smaller the space of possible solutions by inverse $\mathcal{M}'$ the more informative is $\mathcal{M}'$

Example (Recoveries)

- σ: $\{E(x, y)\}$
- τ: $\{F(x, y), G(x)\}$
- $\mathcal{M} = (\sigma, \tau, \Sigma)$ with

$$\Sigma = \{E(x, z) \land E(z, y) \to F(x, y) \land G(z)\}$$

- $\mathcal{M}_1 = (\tau, \sigma, \Sigma_1)$ with

$$\Sigma_1 = \{F(x, y) \to \exists z(E(x, z) \land E(z, y)\}$$

Example (Recoveries)

- σ: $\{E(x, y)\}$
- τ: $\{F(x, y), G(x)\}$
- $\mathcal{M} = (\sigma, \tau, \Sigma)$ with

$$\Sigma = \{E(x, z) \wedge E(z, y) \rightarrow F(x, y) \wedge G(z)\}$$

- $\mathcal{M}_1 = (\tau, \sigma, \Sigma_1)$ with

$$\Sigma_1 = \{F(x, y) \rightarrow \exists z(E(x, z) \wedge E(z, y))\}$$

- $\mathcal{M}_1$ is a recovery of $\mathcal{M}$
  - For any instance $\mathfrak{S}$ let $\mathfrak{T}$ be universal canonical solution for $\mathcal{M}$.
  - Then $(\mathfrak{T}, \mathfrak{S}) \in \mathcal{M}_1$ (so $(\mathfrak{S}, \mathfrak{S}) \in \mathcal{M} \circ \mathcal{M}_1$)

Example (Recoveries)

- σ: $\{E(x,y)\}$
- τ: $\{F(x,y), G(x)\}$
- $\mathcal{M} = (\sigma, \tau, \Sigma)$ with

$$\Sigma = \{E(x,z) \wedge E(z,y) \rightarrow F(x,y) \wedge G(z)\}$$

- $\mathcal{M}_2 = (\tau, \sigma, \Sigma_2)$ with

$$\Sigma_2 = \{G(z) \rightarrow \exists x, y(E(x,z) \wedge E(z,y))\}$$

Example (Recoveries)

- σ: $\{E(x,y)\}$
- τ: $\{F(x,y), G(x)\}$
- $\mathcal{M} = (\sigma, \tau, \Sigma)$ with

$$\Sigma = \{E(x,z) \wedge E(z,y) \rightarrow F(x,y) \wedge G(z)\}$$

- $\mathcal{M}_2 = (\tau, \sigma, \Sigma_2)$ with

$$\Sigma_2 = \{G(z) \rightarrow \exists x, y(E(x,z) \wedge E(z,y))\}$$

- $\mathcal{M}_2$ is a recovery of $\mathcal{M}$

Example (Recoveries)

- σ: $\{E(x, y)\}$
- τ: $\{F(x, y), G(x)\}$
- $\mathcal{M} = (\sigma, \tau, \Sigma)$ with

$$\Sigma = \{E(x, z) \wedge E(z, y) \rightarrow F(x, y) \wedge G(z)\}$$

- $\mathcal{M}_3 = (\tau, \sigma, \Sigma_3)$ with

$$\Sigma_3 = \{F(x, y) \wedge G(z) \rightarrow E(x, z) \wedge E(z, y)\}$$

Example (Recoveries)

- σ: $\{E(x, y)\}$
- τ: $\{F(x, y), G(x)\}$
- $\mathcal{M} = (\sigma, \tau, \Sigma)$ with

$$\Sigma = \{E(x, z) \wedge E(z, y) \rightarrow F(x, y) \wedge G(z)\}$$

- $\mathcal{M}_3 = (\tau, \sigma, \Sigma_3)$ with

$$\Sigma_3 = \{F(x, y) \wedge G(z) \rightarrow E(x, z) \wedge E(z, y)\}$$

- $\mathcal{M}_3$ is not a recovery of $\mathcal{M}$
  - See exercise

Example (Recoveries)

- σ: $\{E(x, y)\}$
- τ: $\{F(x, y), G(x)\}$
- $\mathcal{M} = (\sigma, \tau, \Sigma)$ with

$$\Sigma = \{E(x, z) \wedge E(z, y) \rightarrow F(x, y) \wedge G(z)\}$$

- $\mathcal{M}_4 = (\tau, \sigma, \Sigma_4)$ with

$$\Sigma_4 = \Sigma_1 \cup \Sigma_2$$

Example (Recoveries)

- σ: $\{E(x,y)\}$
- τ: $\{F(x,y), G(x)\}$
- $\mathcal{M} = (\sigma, \tau, \Sigma)$ with

$$\Sigma = \{E(x,z) \wedge E(z,y) \rightarrow F(x,y) \wedge G(z)\}$$

- $\mathcal{M}_4 = (\tau, \sigma, \Sigma_4)$ with

$$\Sigma_4 = \Sigma_1 \cup \Sigma_2$$

- $\mathcal{M}_4$ is a maximum recovery of $\mathcal{M}$
  - can be shown by the following criteria (exercise).

Closure of st-tgds for Maximum Recovery

Proposition

Let $\mathcal{M}'_{\tau\sigma}$ be a recovery of $\mathcal{M}_{\sigma\tau}$. Then $\mathcal{M}'_{\tau\sigma}$ is a maximal recovery iff

1. *For every $(\mathfrak{S}, \mathfrak{S}') \in \mathcal{M} \circ \mathcal{M}'$: $\mathfrak{S}' \in Dom(\mathcal{M})$ and*
2. $\mathcal{M} = \mathcal{M} \circ \mathcal{M}' \circ \mathcal{M}$.

Using this one can show

Theorem

Every mapping specified by a finite set of st-tgds admits a maximum recovery.

Computing Inverses

- Remember algorithms for view rewriting

Proposition

Let $\mathcal{M} = (\sigma, \tau, \Sigma)$ with st-tgds Σ and Q be a CQ over τ.

- *There exists an algorithm QueryRewriting that computes UCQ with equalities Q_{rew} that is a rewriting of Q over the source (i.e. $cert_{\mathcal{M}}(Q, \mathfrak{S}) = Q_{rew}(\mathfrak{S})$ for all source DBs $\mathfrak{S}$).*
- *The algorithm runs in exponential time and its output is of exponential size in the size of Σ, Q.*

- Based on QueryRewriting can define algorithm MaximumRecovery

Theorem

Algorithm MaximumRecovery produces a maximum recovery in exponential time.

Algorithm MaximumRecovery

Input : $\mathcal{M}_{\sigma\tau} = (\sigma, \tau, \Sigma)$ with Σ finite set of st-tgds
Output : A maximum recovery $\mathcal{M}_{\tau\sigma} = (\tau, \sigma, \Gamma)$
$\Gamma := \emptyset$;
forall $\underline{\phi(\vec{x}) \to \exists \vec{y} \psi(\vec{x}, \vec{y}) \in \Sigma}$ **do**
  $Q(\vec{x}) := \exists \vec{y} \psi(\vec{x}, \vec{y})$;
  $\alpha(\vec{x}) := \mathit{QueryRewriting}(\mathcal{M}_{\sigma\tau}, Q)$;
  $\Gamma = \Gamma \cup \{\psi(\vec{x}, \vec{y}) \wedge C(\vec{x}) \to \alpha(\vec{x})\}$;   `//` C `is predicate testing for constant`
end
return $\underline{\mathcal{M}_{\tau\sigma} = (\tau, \sigma, \Gamma)}$;