UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Özgür L. Özçep

Data Exchange 1

Lecture 5: Motivation, Relational DE, Chase
16 November, 2016

Foundations of Ontologies and Databases for Information Systems
CS5130 (Winter 16/17)

Recap of Lecture 4

One of these Lectures ...

- Last lecture was better than the one of last year. Nonetheless, the following video is worth watching:
- `https://www.youtube.com/watch?v=IQgAuBhlBT0`
  Owl video

A Very General Notion of Query

- During the discussion of the reduction of LinORD to CONN we discussed a very general notion of a FOL query. Here is the exact definition. (See Immerman: Descriptive Complexity, p. 18)

Definition

Let τ, σ be any two signatures with $\tau = (R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s)$ and k be a fixed natural number. A **k-ary first order query** $Q : STRUCT(\sigma) \longrightarrow STRUCT(\tau)$ is given by an r+s+1-tuple of σ-formulae $\phi_0, \phi_1, \ldots, \phi_r, \psi_1, \ldots, \psi_s$. For each σ structure $\mathfrak{A} \in STRUC(\sigma)$ the formulae describe a τ structure $Q(\mathfrak{A})$

$$Q(\mathfrak{A}) = (\ dom(Q(\mathfrak{A})), R_1^{Q(\mathfrak{A})}, \ldots, R_r^{Q(\mathfrak{A})}, c_1^{Q(\mathfrak{A})}, \ldots c_s^{Q(\mathfrak{A})}\ )$$

with

- $dom(Q(\mathfrak{A})) = \{(b^1, \ldots, b^k) \mid \mathfrak{A} \models \phi_0(b^1, \ldots, b^k)\}$
- $R_i^{Q(\mathfrak{A})} = \{(b_1^1, \ldots, b_1^k), \ldots, (b_i^1, \ldots, b_i^k) \in dom(Q(\mathfrak{A}))^{a_i} \mid \mathfrak{A} \models \phi_i(b_1^1, \ldots, b_{a_i}^k)\}$
- $c_j^{Q(\mathfrak{A})} =$ the unique $(b^1, \ldots, b^k) \in dom(Q(\mathfrak{A}))$ s.t. $\mathfrak{A} \models \psi_j(b^1, \ldots, b^k)$

Example: Reductio of linear order to connectivity
$Q_{red} : LinOrd \rightarrow CONN$

- $\tau = E$, $\sigma =<$, $r = 1$, $s = 0$
- $k = 1$, $\phi_0 =$ an arbitrary tautology
- $\phi_1 =$ see Exercise 3.3

- **Locality** as a means for proving in-expressivity results for logics
  - Hanf Locality
    Answers are the same on two structures which are point-wise similar (Ex. 4.1)
  - Gaifman locality
    Query cannot distinguish between tuples which are locally the same in the given structure
  - Bounded number of Degree (BNDP)
    Cannot produce more degrees in output w.r.t. a given bound than in the input
  - Relations: Hanf $\models$ Gaifman $\models$ BNDP
- 0-1 law
  Almost all structures have property or almost all have not property.
- 0-1 law works also for logics with recursion (Datalog) (Ex. 4.3)

End of Recap

Data Exchange: Motivation

References

- (Arenas et al., 2014)
  M. Arenas, P. Barceló, L. Libkin, and F. Murlak. Foundations of Data Exchange. Cambridge University Press, 2014.
- M. Arenas: Slides to "Data Exchange in the Relational and RDF Worlds", Fifth Workshop on Semantic Web Information Management 2011

Data Exchange History

- Much research in DB community
- Incorporated into IBM Clio
- Formal treatment starts with 2003 paper by Fagin and colleagues

  Lit: R. Fagin, L. M. Haas, M. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Conceptual modeling: Foundations and applications. chapter Clio: Schema Mapping Creation and Data Exchange, pages 198–236. Springer-Verlag, Berlin, Heidelberg, 2009.

  Lit: R. Fagin et al. Data exchange: Semantics and query answering. In: Database Theory - ICDT 2003, 2003, Proceedings, volume 2572 of LNCS, pages 207–224. Springer, 2003.

Semantic Integration

- Data Exchange a form of semantic integration

- Research area **semantic integration (SI)**
  Deals with **issues** related to ensuring **interoperability** of possibly **heterogeneous data sources**.

- Lecture 5 and 6: Data Exchange: Directed DB-level SI for source and target DB
- Following lectures
  - OBDA: Bridging the DB and ontology world
  - Ontology-level integration

Data Exchange (DE)

- DE deals in a specific way with the integration of DBs
- Heterogeneity: Two DBs on the same domain but different schemata, σ (**source**) and τ (**target**)
- Interoperability: Relationship specifications $M_{\tau\sigma}$ for σ and τ
- Relevant service: **Query answering** over τ
- Challenges
  - **Consistency:** Is there a corresponding τ instance for a given σ instance?
  - **Materialization:** If yes, construct and materialize exactly one instance for τ
  - **Query answering:** Answer query on this instance (using rewriting)
  - **Maintenance:** How to construct/maintain mappings

Data Exchange (DE)

- DE deals in a specific way with the integration of DBs
- Heterogeneity: Two DBs on the same domain but different schemata, σ (**source**) and τ (**target**)
- Interoperability: Relationship specifications $M_{\tau\sigma}$ for σ and τ
- Relevant service: **Query answering** over τ
- Challenges
  - **Consistency:** Is there a corresponding τ instance for a given σ instance?
  - **Materialization:** If yes, construct and materialize exactly one instance for τ
  - **Query answering:** Answer query on this instance (using rewriting)
  - **Maintenance:** How to construct/maintain mappings

Relational DE

- Going to deal mainly with relational DBs

- Language for specifying $M_{\sigma\tau}$: Specific FOL formulas called tuple generating dependencies (tgds)
- Allow for constraints on the target schema (such as foreign keys)

- Explicate criteria for goodness of solutions by **universal model** and **core** notion

- Query answering w.r.t. **certain answer semantics** and using rewriting

Running Example: Flight Domain

Source schema σ

Geo(city, coun, pop)

Flight (src, dest, airl, dep)

Target DB τ

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

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

Serves(airl, city, coun, phone)

- Instead of changing the source schema σ, invent own (target) schema τ
- Query over target schema

Running Example: Flight Domain

Source schema σ

Geo(city, coun, pop)

Flight (src, dest, airl, dep)

Target DB τ

Routes(fno, src, dest)

Info(fno, dep, arr, airl)

Serves(airl, city, coun, phone)

- Find "corresponding" τ DB instances for given σ instances
- Correspondence ensured by **mapping rules** $M_{\sigma\tau}$
  1. $\mathit{Flight}(\mathit{src}, \mathit{dest}, \mathit{airl}, \mathit{dep}) \longrightarrow$
     $\exists \mathit{fno}\ \exists\ \mathit{arr}(\mathit{Routes}(\mathit{fno}, \mathit{src}, \mathit{dest}) \wedge \mathit{Info}(\mathit{fno}, \mathit{dep}, \mathit{arr}, \mathit{airl}))$
  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. $\mathit{Flight}(\mathit{src}, \mathit{city}, \mathit{airl}, \mathit{dep}) \wedge \mathit{Geo}(\mathit{city}, \mathit{coun}, \mathit{pop}) \longrightarrow$
     $\exists \mathit{phone}\ (\mathit{Serves}(\mathit{airl}, \mathit{city}, \mathit{coun}, \mathit{phone}))$

Running Example: Flight Domain

Source schema σ and instance

Geo(	city,	coun,	pop	)
Flight (	src,	dest,	airl,	dep)
	paris	sant.	airFr	2320

Target DB τ

Routes(fno, src, dest)

Info(fno, dep, arr, airl)

Serves(airl, city, coun, phone)

- Find "corresponding" τ DB instances for given σ instances
- Correspondence ensured by **mapping rules** $M_{\sigma\tau}$
  1. $Flight(src, dest, airl, dep) \longrightarrow$
     $\exists fno \; \exists \; arr(Routes(fno, src, dest) \wedge Info(fno, dep, arr, airl))$
  2. $Flight(city, dest, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow$
     $\exists phone \; (Serves(airl, city, coun, phone))$
  3. $Flight(src, city, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow$
     $\exists phone \; (Serves(airl, city, coun, phone))$

Running Example: Flight Domain

Source schema σ and instance

Geo(	city,	coun,	pop	)

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

Target DB τ

Routes(fno, src, dest)

Info(fno, dep, arr, airl)

Serves(airl, city, coun, phone)

- Find "corresponding" τ DB instances for given σ instances
- Correspondence ensured by **mapping rules** $M_{\sigma\tau}$

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

Running Example: Flight Domain

Source schema σ and instance

Geo(	city,	coun,	pop	)
Flight (	src,	dest,	airl,	dep)
	paris	sant.	airFr	2320

Target DB τ and instance

Routes(	fno,	src,	dest		)
	$\bot_1$,	paris,	sant.		
Info(	fno,	dep,	arr,	airl	)
	$\bot_1$,	2320,	$\bot_2$	airFr	
Serves(	airl,	city,	coun,	phone	)

- Find "corresponding" τ DB instances for given σ instances
- Correspondence ensured by **mapping rules** $M_{\sigma\tau}$

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

2. $Flight(city, dest, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow$
   $\exists phone\ (Serves(airl, city, coun, phone))$

3. $Flight(src, city, airl, dep) \wedge Geo(city, coun, pop) \longrightarrow$
   $\exists phone\ (Serves(airl, city, coun, phone))$

Running Example: Flight Domain

Source schema σ and instance

Geo(city, coun, pop)

Flight (src, dest, airl, dep)

src	dest	airl	dep
paris	sant.	airFr	2320

Target DB τ and instance

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

<u>fno</u>	src	dest
$\bot_1$,	paris,	sant.

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

<u>fno</u>	dep	arr	airl
$\bot_1$,	2320,	$\bot_2$	airFr

Serves(airl, city, coun, phone)

- σ-instance

$$\mathfrak{S} = \{Flight(paris, sant, airFr, 2320)\}$$

- τ **solution**

$$\mathfrak{T} = \{Routes(\bot_1, paris, sant), Info(\bot_1, 2320, \bot_2, airFr)\}$$

- In general there may be more than one solution:

$$\mathfrak{T}' = \{Routes(123, paris, sant), Info(123, 2320, \bot_2, airFr)\}$$

- Have to answer queries w.r.t. all solutions: **certain answers**

Running Example: Flight Domain

Source schema σ and instance

Geo(city, coun, pop)

Flight (src, dest, airl, dep)

Target DB τ and instance

Routes(fno, src, dest)

Info(fno, dep, arr, airl)

Serves(airl, city, coun, phone)

- σ-instance
$$\mathfrak{S} = \{Flight(paris, sant, airFr, 2320)$$
- Boolean query $Q_1 = \exists fno\ Routes(fno, paris, sant)$
  - Certain answers is yes, because in all solutions there is a route form Paris to Santiago
- Boolean query $Q_2 = Routes(123, paris, sant)$
  - Certain answer is no

Relational Mappings

- Going to deal mainly with relational mappings
- Relational DB (Codd 1970) very successful and still highly relevant
- There were other opinions...

"Some of the ideas presented in the paper are interesting and may be of some use, but, in general, this very preliminary work fails to make a convincing point as to their implementation, performance, and practical usefulness. The paper's general point is that the tabular form presented should be suitable for general data access, but I see two problems with this statement: expressivity and efficiency. [...] The formalism is needlessly complex and mathematical, using concepts and notation with which the average data bank practitioner is unfamiliar." Cited according to (Santini 2005)

Lit: E. F. Codd. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387, June 1970.

Lit: S. Santini. We are sorry to inform you ... Computer, December 2005.

Relational Mappings Formally

Definition

A **relational mapping** $\mathcal{M}$ is a tuple of the form

$$\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_{\tau})$$

where

- σ is the source schema
- τ is the target schema with all relation symbols different from those in σ
- $M_{\sigma\tau}$ is a finite set of FOL formulae over $\sigma \cup \tau$ called source-to-target dependencies
- M_{τ} is a set of constraints on the target schema called target dependencies

DB Instances of Schemata

- Schemata are relational signatures
- **Concrete database instance**
  - For a given schema σ a concrete DB instance is a σ FOL structure with active domain
  - **Active domain:** Domain contains all and only individuals (also called constants) occurring in relations
  - Usually: All source instances are concrete DBs
- **Generalized DB instances**
  - For some attributes in target schema (Example: flight number fno) no corresponding attribute in source may exist
  - Next to constants CONST allow disjoint set of marked NULLs, denoted VAR
  - A generalized DB instance may contain elements from CONST $\cup$ VAR

DB Instances of Schemata

- Schemata are relational signatures
- **Concrete database instance**
  - For a given schema σ a concrete DB instance is a σ FOL structure with active domain
  - **Active domain:** Domain contains all and only individuals (also called constants) occurring in relations
  - Usually: All source instances are concrete DBs
- **Generalized DB instances**
  - For some attributes in target schema (Example: flight number fno) no corresponding attribute in source may exist
  - Next to constants CONST allow disjoint set of marked NULLs, denoted VAR
  - A generalized DB instance may contain elements from CONST $\cup$ VAR

Source-Target-Dependencies $M_{\sigma\tau}$

- Source-Target-Dependencies may be arbitrary FOL formula
- But usually they have a simple directed form
  - required to ensure decidability
- Here: source-to-target tuple-generating dependencies (st-tgds)

Definition

A **source-to-target tuple-generating dependencies (st-tgds)** is a FOL formula of the form

$$\forall \vec{x}\vec{y}(\phi_\sigma(\vec{x},\vec{y}) \longrightarrow \exists \vec{z}\, \psi_\tau(\vec{x},\vec{z}))$$

where

- ϕ_σ is a conjunction of atoms over source schema σ
- ψ_τ is a conjunction of atoms over target schema τ

Reminder: Conjunctive Queries (CQs)

- Class of sufficiently expressive and feasible FOL queries of 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
  - $\alpha_i(\vec{x_i}, \vec{y_i})$ are atomic FOL formula and
  - $\vec{x_i}$ variable vectors among $\vec{x}$ and $\vec{y_i}$ variables among $\vec{y}$
- Corresponds to SELECT-PROJECT-JOIN Fragment of SQL

Reminder: Conjunctive Queries (CQs)

Theorem

- *Answering CQs is NP-complete w.r.t. combined complexity (Chandra,Merlin 1977)*
- *Subsumption test for CQs is NP complete*
- *Answering CQs is in AC^0 (and thus in P) w.r.t. data complexity*

Lit: A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In: Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, STOC'77, pages 77–90, New York, NY, USA, 1977. ACM.

Wake-Up Question

Are st-tgds Datalog rules?

Wake-Up Question

Are st-tgds Datalog rules?

- No, as Datalog rules do not allow existentials in the head of the query
- But there is the extended logic called Datalog$+/-$
  - Has been investigated in last years also in context of ontology-based data access (see net lectures)
  - Provides many interesting sub-fragments

Lit: A. Calì, G. Gottlob, and T. Lukasiewicz. Datalog+/-: A unified approach to ontologies and integrity constraints. In Proceedings of the 12th International Conference on Database Theory, pages 14–30. ACM Press, 2009.

Target Dependencies M_τ

- These define constraints on target schema known also from classical DB theory
- Two different types of dependencies are sufficiently general to capture the classical DB constraints

Definition

A **tuple-generating dependency (tgd)** is a FOL formula of the form

$$\forall \vec{x}\vec{y}(\phi(\vec{x},\vec{y}) \longrightarrow \exists \vec{z}\, \psi(\vec{x},\vec{z}))$$

where ϕ, ψ are conjunctions of atoms over τ.

An **equality-generating (egd)** is a FOL formula of the form

$$\forall \vec{x}(\phi(\vec{x}) \longrightarrow x_i = x_j)$$

where $\phi(\vec{x})$ is a conjunction of atoms over τ and x_i, x_j occur in $\vec{x}$.

Semantics: Solutions

Definition

Given: a mapping $\mathcal{M}$ and a σ instance $\mathfrak{S}$

A τ instance $\mathfrak{T}$ is called a **solution** for $\mathfrak{S}$ under $\mathcal{M}$ iff $(\mathfrak{S}, \mathfrak{T})$ satisfies all rules in $M_{\sigma\tau}$ (for short: $(\mathfrak{S}, \mathfrak{T}) \models M_{\sigma\tau}$) and $\mathfrak{T}$ satisfies all rules in M_{τ}.

- $(\mathfrak{S}, \mathfrak{T}) \models M_{\sigma\tau}$ iff $\mathfrak{S} \cup \mathfrak{T} \models M_{\sigma\tau}$ where
  - $\mathfrak{S} \cup \mathfrak{T}$ is the union of the instances $\mathfrak{S}, \mathfrak{T}$: Structure containing all relations from $\mathfrak{S}$ and $\mathfrak{T}$ with domain the union of domains of $\mathfrak{S}$ and $\mathfrak{T}$
  - well defined because schemata are disjoint
- $Sol_{\mathcal{M}}(\mathfrak{S})$: Set of solutions for $\mathfrak{S}$ under $\mathcal{M}$

First Key Problem: Existence of Solutions

Problem: SOLEXISTENCE$_{\mathcal{M}}$

Input: Source instance $\mathfrak{S}$
Output: Answer whether there exists a solution for $\mathfrak{S}$ under $\mathcal{M}$

- Note: $\mathcal{M}$ is assumed to be fixed $\Longrightarrow$ data complexity
- This problem is going to be approached with a well known proof tool: chase

Trivial Case: No Target Dependencies

- Without target constraints there is always a solution

Proposition

Let $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau})$ with $M_{\sigma\tau}$ consisting of st-tgds. Then for any source instance $\mathfrak{S}$ there are infinitely many solutions and at least one solution can be constructed in polynomial time.

Trivial Case: No Target Dependencies

- Without target constraints there is always a solution

Proposition

Let $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau})$ with $M_{\sigma\tau}$ consisting of st-tgds. Then for any source instance $\mathfrak{S}$ there are infinitely many solutions and at least one solution can be constructed in polynomial time.

Proof Idea

- For every rule and every tuple $\vec{a}$ fulfilling the head generate facts according to the body (using fresh named nulls for the existentially quantified variables)
- Resulting τ instance $\mathfrak{T}$ is a solution
- Polynomial: Testing whether $\vec{a}$ fulfills the head (a conjunctive query) can be done in polynomial time
- Infinity: From $\mathfrak{T}$ can build any other solution by extension

Undecidability for General Constraints

Theorem

There is a relational mapping $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$ such that $\mathit{SOLEXISTENCE}_{\mathcal{M}}$ *is undecidable.*

- Proof by reduction from embedding problem for finite semigroups which is known to be undecidable (Arenas et al. 2014, Thm 5.3)
- As a consequence: Further restrict mapping rules
- But note that the following chase construction defined for arbitrary st-tgds

Undecidability for General Constraints

Theorem

There is a relational mapping $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$ such that $SOLEXISTENCE_{\mathcal{M}}$ is undecidable.

Wake-Up Question

As another exercise in reduction prove the following corollary:
There is a relational mapping $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau})$ with a single FOL dependency in $M_{\sigma\tau}$ s.t. $\mathrm{SOLEXISTENCE}_{\mathcal{M}}$ is undecidable

Undecidability for General Constraints

Theorem

There is a relational mapping $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$ such that SOLEXISTENCE$_\mathcal{M}$ is undecidable.

Wake-Up Question

As another exercise in reduction prove the following corollary:
There is a relational mapping $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau})$ with a single FOL dependency in $M_{\sigma\tau}$ s.t. SOLEXISTENCE$_\mathcal{M}$ is undecidable

Proof

- Assume otherwise
- Given $\mathcal{M} = (\sigma, \tau, M_{\sigma\tau}, M_\tau)$
- construct $\mathcal{M}' = (\sigma, \tau, \{\chi\})$ with
- $\chi = \bigwedge(M_{\sigma\tau} \cup M_\tau)$

Existence Proof vs. Construction

- Proposition above showed existence of solution
- Showing existence $\neq$ construction a verifier
- Actually we are going to construct a solution using the chase

- Interesting debate in philosophy of mathematics whether non-constructive proofs are acceptable
- **Mathematical Intuitionism:** field allowing only constructive proofs
  - truth = provable = constructively provable
  - Classical logical inference rules s.a. $\neg\neg A \models A$ not allowed
  - Main inventor: L.E.J. Brouwer (1881 to 1966)
    Irony: Has many interesting results in classical (non-constructive) mathematics (Brouwer's fixed point theorem)

Chase Construction

- A widely used tool in DB theory
- Original use: Calculating entailments of DB constraints

  Lit: D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. ACM Trans. Database Syst., 4(4):455–469, Dec. 1979.

- **General idea**
  - Apply tgds as completion/repair rules in a bottom-up strategy
  - until no tgds can be applied anymore
  - Chase construction mail fail if one of the egds is violated

- The chase leads to an instance with desirable properties
  - It produces not too many redundant facts
  - Universality

Example (Terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x, y) \to G(x, y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x, y) \to \exists z\, L(y, z)}_{\chi_1} \}$
- Source instance $\mathfrak{S} = \{E(a, b)\}$
- Going to build stepwise potential target instances $\mathfrak{T}_i$ considering pairs $(\mathfrak{S}, \mathfrak{T}_i)$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a, b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a, b), L(b, \bot)\})$ (termination)

Example (Terminating c(h)ase)

- Source schema $\sigma = \{E\}$;   target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\, L(y,z)}_{\chi_1} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$
- Going to build stepwise potential target instances $\mathfrak{T}_i$ considering pairs $(\mathfrak{S}, \mathfrak{T}_i)$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp)\})$ (termination)

Example (Terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\, L(y,z)}_{\chi_1} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$
- Going to build stepwise potential target instances $\mathfrak{T}_i$ considering pairs $(\mathfrak{S}, \mathfrak{T}_i)$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp)\})$ (termination)

Example (Terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \to G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \to \exists z\, L(y,z)}_{\chi_1} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$
- Going to build stepwise potential target instances $\mathfrak{T}_i$ considering pairs $(\mathfrak{S}, \mathfrak{T}_i)$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp)\})$ (termination)

Example (Terminating c(h)ase)

- Source schema $\sigma = \{E\}$;   target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \to G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \to \exists z\, L(y,z)}_{\chi_1} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$
- Going to build stepwise potential target instances $\mathfrak{T}_i$ considering pairs $(\mathfrak{S}, \mathfrak{T}_i)$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp)\})$ (termination)

Example (Terminating c(h)ase)

- Source schema $\sigma = \{E\}$;   target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\, L(y,z)}_{\chi_1} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$
- Going to build stepwise potential target instances $\mathfrak{T}_i$ considering pairs $(\mathfrak{S}, \mathfrak{T}_i)$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot)\})$ (termination)

Example (Non-terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\ L(y,z)}_{\chi_1}, \underbrace{L(x,y) \rightarrow \exists z\ G(y,z)}_{\chi_2} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp)\})$ (violates χ_2)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp), G(\perp,\perp_1)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp), G(\perp,\perp_1), L(\perp_1,\perp_2)\})$ (violates χ_2)
- ... (non-termination)

Example (Non-terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\ L(y,z)}_{\chi_1}, \underbrace{L(x,y) \rightarrow \exists z\ G(y,z)}_{\chi_2} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp)\})$ (violates χ_2)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp), G(\perp,\perp_1)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp), G(\perp,\perp_1), L(\perp_1,\perp_2)\})$ (violates χ_2)
- ... (non-termination)

Example (Non-terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\ L(y,z)}_{\chi_1}, \underbrace{L(x,y) \rightarrow \exists z\ G(y,z)}_{\chi_2} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot)\})$ (violates χ_2)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot), G(\bot,\bot_1)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot), G(\bot,\bot_1), L(\bot_1,\bot_2)\})$ (violates χ_2)
- ... (non-termination)

Example (Non-terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \to G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \to \exists z\, L(y,z)}_{\chi_1}, \underbrace{L(x,y) \to \exists z\, G(y,z)}_{\chi_2} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot)\})$ (violates χ_2)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot), G(\bot,\bot_1)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot), G(\bot,\bot_1), L(\bot_1,\bot_2)\})$ (violates χ_2)
- ... (non-termination)

Example (Non-terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\, L(y,z)}_{\chi_1}, \underbrace{L(x,y) \rightarrow \exists z\, G(y,z)}_{\chi_2} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp)\})$ (violates χ_2)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp), G(\perp,\perp_1)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp), G(\perp,\perp_1), L(\perp_1,\perp_2)\})$ (violates χ_2)
- ... (non-termination)

Example (Non-terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\, L(y,z)}_{\chi_1}, \underbrace{L(x,y) \rightarrow \exists z\, G(y,z)}_{\chi_2} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot)\})$ (violates χ_2)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot), G(\bot,\bot_1)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot), G(\bot,\bot_1), L(\bot_1,\bot_2)\})$ (violates χ_2)
- ... (non-termination)

Example (Non-terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\ L(y,z)}_{\chi_1}, \underbrace{L(x,y) \rightarrow \exists z\ G(y,z)}_{\chi_2} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot)\})$ (violates χ_2)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot), G(\bot,\bot_1)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot), G(\bot,\bot_1), L(\bot_1,\bot_2)\})$ (violates χ_2)
- ... (non-termination)

Example (Non-terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\ L(y,z)}_{\chi_1}, \underbrace{L(x,y) \rightarrow \exists z\ G(y,z)}_{\chi_2} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp)\})$ (violates χ_2)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp), G(\perp,\perp_1)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\perp), G(\perp,\perp_1), L(\perp_1,\perp_2)\})$ (violates χ_2)
- ... (non-termination)

Example (Non-terminating c(h)ase)

- Source schema $\sigma = \{E\}$; target schema $\tau = \{G, L\}$
- $M_{\sigma\tau} = \{ \underbrace{E(x,y) \rightarrow G(x,y)}_{\theta_1} \}$

  $M_{\tau} = \{ \underbrace{G(x,y) \rightarrow \exists z\, L(y,z)}_{\chi_1}, \underbrace{L(x,y) \rightarrow \exists z\, G(y,z)}_{\chi_2} \}$
- Source instance $\mathfrak{S} = \{E(a,b)\}$

- $(\mathfrak{S}, \emptyset)$ (violates θ_1)
- $(\mathfrak{S}, \{G(a,b)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot)\})$ (violates χ_2)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot), G(\bot,\bot_1)\})$ (violates χ_1)
- $(\mathfrak{S}, \{G(a,b), L(b,\bot), G(\bot,\bot_1), L(\bot_1,\bot_2)\})$ (violates χ_2)
- ... (non-termination)

Chase Definition

- Let $\mathfrak{S}$ be a σ instance and *dom*($\mathfrak{S}$) its domain

Definition (Chase steps)

$\boxed{\mathfrak{S} \overset{\chi,\vec{a}}{\rightsquigarrow} \mathfrak{S}'}$ iff

1. χ a **tgd** of form $\phi(\vec{x}) \rightarrow \exists \vec{y} \psi(\vec{x}, \vec{y})$ and
   - $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from *dom*($\mathfrak{S}$)
   - $\mathfrak{S}'$ extends $\mathfrak{S}$ with all atoms occurring in $\psi(\vec{a}, \vec{\bot})$.
2. or χ is an **egd** of form $\phi(\vec{x}) \rightarrow x_i = x_j$ and
   - $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from *dom*($\mathfrak{S}$) with $a_i \neq a_j$ and
   - (a_i is constant or null, a_j is null and $\mathfrak{S}' = \mathfrak{S}[a_j/a_i]$ or
   - a_i is null, a_j is constant and $\mathfrak{S}' = \mathfrak{S}[a_i/a_j]$)

$\boxed{\mathfrak{S} \overset{\chi,\vec{a}}{\rightsquigarrow} \textit{fail}}$ iff

- $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from *dom*($\mathfrak{S}$) with $a_i \neq a_j$
- and both a_i, a_j are constants.

Chase Definition

- Let $\mathfrak{S}$ be a σ instance and $dom(\mathfrak{S})$ its domain

Definition (Chase steps)

$\boxed{\mathfrak{S} \overset{\chi,\vec{a}}{\rightsquigarrow} \mathfrak{S}'}$ iff

1. χ a **tgd** of form $\phi(\vec{x}) \rightarrow \exists \vec{y} \psi(\vec{x}, \vec{y})$ and
   - $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from $dom(\mathfrak{S})$
   - $\mathfrak{S}'$ extends $\mathfrak{S}$ with all atoms occurring in $\psi(\vec{a}, \vec{\bot})$.
2. or χ is an **egd** of form $\phi(\vec{x}) \rightarrow x_i = x_j$ and
   - $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from $dom(\mathfrak{S})$ with $a_i \neq a_j$ and
   - (a_i is constant or null, a_j is null and $\mathfrak{S}' = \mathfrak{S}[a_j/a_i]$ or
   - a_i is null, a_j is constant and $\mathfrak{S}' = \mathfrak{S}[a_i/a_j]$)

$\boxed{\mathfrak{S} \overset{\chi,\vec{a}}{\rightsquigarrow} \mathit{fail}}$ iff

- $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from $dom(\mathfrak{S})$ with $a_i \neq a_j$
- and both a_i, a_j are constants.

Chase Definition

- Let $\mathfrak{S}$ be a σ instance and $dom(\mathfrak{S})$ its domain

Definition (Chase steps)

$\boxed{\mathfrak{S} \overset{\chi,\vec{a}}{\rightsquigarrow} \mathfrak{S}'}$ iff

1. χ a **tgd** of form $\phi(\vec{x}) \rightarrow \exists \vec{y} \psi(\vec{x}, \vec{y})$ and
   - $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from $dom(\mathfrak{S})$
   - $\mathfrak{S}'$ extends $\mathfrak{S}$ with all atoms occurring in $\psi(\vec{a}, \vec{\bot})$.
2. or χ is an **egd** of form $\phi(\vec{x}) \rightarrow x_i = x_j$ and
   - $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from $dom(\mathfrak{S})$ with $a_i \neq a_j$ and
   - (a_i is constant or null, a_j is null and $\mathfrak{S}' = \mathfrak{S}[a_j/a_i]$ or
   - a_i is null, a_j is constant and $\mathfrak{S}' = \mathfrak{S}[a_i/a_j]$)

$\boxed{\mathfrak{S} \overset{\chi,\vec{a}}{\rightsquigarrow} \textit{fail}}$ iff

- $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from $dom(\mathfrak{S})$ with $a_i \neq a_j$
- and both a_i, a_j are constants.

Chase Definition

- Let $\mathfrak{S}$ be a σ instance and $dom(\mathfrak{S})$ its domain

Definition (Chase steps)

$\boxed{\mathfrak{S} \overset{\chi,\vec{a}}{\rightsquigarrow} \mathfrak{S}'}$ iff

1. χ a **tgd** of form $\phi(\vec{x}) \rightarrow \exists \vec{y} \psi(\vec{x}, \vec{y})$ and
   - $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from $dom(\mathfrak{S})$
   - $\mathfrak{S}'$ extends $\mathfrak{S}$ with all atoms occurring in $\psi(\vec{a}, \vec{\bot})$.
2. or χ is an **egd** of form $\phi(\vec{x}) \rightarrow x_i = x_j$ and
   - $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from $dom(\mathfrak{S})$ with $a_i \neq a_j$ and
   - (a_i is constant or null, a_j is null and $\mathfrak{S}' = \mathfrak{S}[a_j/a_i]$ or
   - a_i is null, a_j is constant and $\mathfrak{S}' = \mathfrak{S}[a_i/a_j]$)

$\boxed{\mathfrak{S} \overset{\chi,\vec{a}}{\rightsquigarrow} \mathit{fail}}$ iff

- $\mathfrak{S} \models \phi(\vec{a})$ for some elements $\vec{a}$ from $dom(\mathfrak{S})$ with $a_i \neq a_j$
- and both a_i, a_j are constants.

Chase

Definition

A **chase sequence for $\mathfrak{S}$ under** M is a sequence of chase steps $\mathfrak{S}_i \overset{\chi_i, \vec{a_i}}{\rightsquigarrow} \mathfrak{S}_{i+1}$ such that

- $\mathfrak{S}_0 = \mathfrak{S}$
- each χ_i is in M
- for each distinct i, j also $(\chi_i, \vec{a_i}) \neq (\chi_j, \vec{a_j})$

For a finite chase sequence the last instance is called its **result**.

- If the result is *fail*, then the sequence is said to be a **failing sequence**
- If no further dependency from M can be applied to a result, then the sequence is called **successful**.

Indeterminism

- Indeterminism regarding choice of nulls (no problem)
- Indeterminism regarding order of chosen tgds and egds

  This may lead to different chase results

Use of Chases in Data Exchange

- A chase sequence for $\mathfrak{S}$ under a $\mathcal{M}$ is a chase sequence for $(\mathfrak{S}, \emptyset)$ under $M_{\sigma\tau} \cup M_{\tau}$
- If $(\mathfrak{S}, \mathfrak{T})$ result of a finite sequence, call just $\mathfrak{T}$ the result

- Chase is the right tool for finding solutions

Proposition

Given $\mathcal{M}$ and source instance $\mathfrak{S}$.

- *If there is a successful chase sequence for $\mathfrak{S}$ with result $\mathfrak{T}$, then $\mathfrak{T}$ is a solution.*
- *If there is a failing chase sequence for $\mathfrak{S}$, then $\mathfrak{S}$ has no solution.*

Use of Chases in Data Exchange

- A chase sequence for $\mathfrak{S}$ under a $\mathcal{M}$ is a chase sequence for $(\mathfrak{S}, \emptyset)$ under $M_{\sigma\tau} \cup M_{\tau}$
- If $(\mathfrak{S}, \mathfrak{T})$ result of a finite sequence, call just $\mathfrak{T}$ the result

- Chase is the right tool for finding solutions

Proposition

Given $\mathcal{M}$ and source instance $\mathfrak{S}$.

- *If there is a successful chase sequence for $\mathfrak{S}$ with result $\mathfrak{T}$, then $\mathfrak{T}$ is a solution.*
- *If there is a failing chase sequence for $\mathfrak{S}$, then $\mathfrak{S}$ has no solution.*

- The proposition does no cover all cases: non-terminating chase
- In this case still there still may be a solution

Weak Acyclicity

- In order to guarantee termination restrict target constraints
- Reason for non-termination: generation of new nulls with same dependencies

Example (Cycle in Dependencies)

- $\chi_1 = G(x, y) \rightarrow \exists z\ L(y, z)$
- $\chi_2 = L(x, y) \rightarrow \exists z\ G(y, z)$

Possible infinite generation

$$G(a, b) \overset{\chi_1}{\rightsquigarrow} L(b, \perp_1) \overset{\chi_2}{\rightsquigarrow} G(\perp_1, \perp_2) \overset{\chi_1}{\rightsquigarrow} L(\perp_2, \perp_3) \dots$$

- Problem caused by cycle in dependencies

Weak Acyclicity

- In order to guarantee termination restrict target constraints
- Reason for non-termination: generation of new nulls with same dependencies

Example (Cycle in Dependencies)

- $\chi_1 = G(x, y) \rightarrow \exists z \; L(y, z)$
- $\chi_2 = L(x, y) \rightarrow \exists z \; G(y, z)$

Possible infinite generation

$$G(a, b) \overset{\chi_1}{\rightsquigarrow} L(b, \bot_1) \overset{\chi_2}{\rightsquigarrow} G(\bot_1, \bot_2) \overset{\chi_1}{\rightsquigarrow} L(\bot_2, \bot_3) \ldots$$

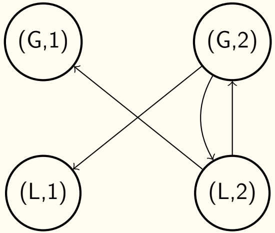
- Problem caused by cycle in dependencies

Simple Dependency Graphs

- Nodes: pairs (R, i) of predicate R and argument-position i
- Edges: From (R_b, i) to (R_h, j) iff there is a tgd $\forall\vec{x}\forall\vec{y}\phi(\vec{x}, \vec{y}) \rightarrow \exists\vec{z}\psi(\vec{x}, \vec{z})$ and
  1. R_h occurs in ψ and R_b occurs in ϕ and
  2. for all $x \in \vec{x}$ in i-position in R_b
     - either x occurs in j-position in R_h
     - or the variable in j-position in R_h is existentially quantified

Example (Simple Dependency Graph)

- $\chi_1 = G(y, x) \rightarrow \exists z\ L(x, z)$
- $\chi_2 = L(y, x) \rightarrow \exists z\ G(x, z)$



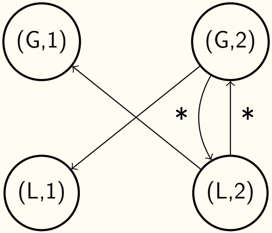
Set of tgds called **acyclic** if simple dependency graph is acyclic.

Dependency Graphs (DG)

- Nodes: pairs (R, i) of predicate R and argument-position i
- Edges: From (R_b, i) to (R_h, j) iff there is a tgd $\forall \vec{x} \forall \vec{y} \phi(\vec{x}, \vec{y}) \rightarrow \exists \vec{z} \psi(\vec{x}, \vec{z})$ and
  1. R_h occurs in ψ and R_b occurs in ϕ and
  2. for all $x \in \vec{x}$ in i-position in R_b
     - either x occurs in j-position in R_h
     - or the variable in j-position in R_h is existentially quantified and and these are labelled by *

Example (Dependency Graph)

- $\chi_1 = G(y, x) \rightarrow \exists z\ L(x, z)$
- $\chi_2 = L(y, x) \rightarrow \exists z\ G(x, z)$



TGDs **weakly acyclic** iff DG has no cycle with a * edge.

Termination for weakly acyclic tgds

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. Then the length of every chase sequence for a source $\mathfrak{S}$ is polynomially bounded w.r.t. the size of $\mathfrak{S}$.

- In particular: Every chase sequence terminates
- Moreover: $\text{SOLEXISTENCE}_{\mathcal{M}}$ can be solved in polynomial time
- a solution can be constructed in polynomial time

Solutions to Exercise 4 (16 Points)

Solution to Exercise 4.1 (6 Points)

Use Hanf locality in order to proof that the following boolean queries are not FOL-definable: 1. graph acyclicity, 2. tree.

Solution

Graph Acyclicity (GA).

- For contradiction assume GA is Hanf-local with parameter r'. Choose $r = 2r' + 2$
- Let $\mathfrak{G}$ be the disjoint union of a circle of length r and a linear order of length r
- Let $\mathfrak{G}'$ be an order of length $2r$.
- Take a bijection $f : \mathfrak{G} \to \mathfrak{G}'$ where
  - the circle is unravelled to the middle of $\mathfrak{G}'$.
  - The lower half part of the order in $\mathfrak{G}$ is mapped to the lower part of $\mathfrak{G}'$
  - The upper half part of the order in $\mathfrak{G}$ is mapped to the upper part of $\mathfrak{G}'$
- an r'-neighbourhood of any a in $\mathfrak{G}$ and $f(a) \in \mathfrak{G}'$ is the same: if a is from the circle in $\mathfrak{G}$ then the r'-neighbourhood is a 2r'-line and the same for $f(a)$. If a is an element from the line in $\mathfrak{G}$ then in its r'-neighbourhood it has to the left and to right the same number of elements as has $f(a)$ in its r'-neigbourhood in $\mathfrak{G}'$.
- Hence $\mathfrak{G} \rightleftarrows_{r'} \mathfrak{G}'$, but: $\mathfrak{G}$ is cyclic and $\mathfrak{G}$ is not. ↯

Tree

- Same construction (as $\mathfrak{G}'$ is tree whereas $\mathfrak{G}$ is not)

Solution to Exercise 4.2 (4 Points)

Show that $EVEN(\sigma)$ can be defined within second-order logic for any σ.

Hint: formalize "There is a binary relation which is an equivalence relation having only equivalence classes with exactly two elements" and argue why this shows the axiomatizability.

Solution

$$
\begin{aligned}
\exists R \qquad & \forall x R(x,x) \land \\
& \forall x \forall y R(x,y) \rightarrow R(y,x) \land \\
& \forall x \forall y \forall z ((R(x,y) \land R(y,z)) \rightarrow R(x,z)) \land \\
& \forall x \exists y (R(x,y) \land x \neq y \land \forall z (R(x,z) \rightarrow z = x \lor z = y))
\end{aligned}
$$

Note that R is a quantified variable (!). So we have shown that $EVEN[\emptyset]$ can be defined.

Solution to Exercise 4.3 (2 Points)

Argue why (in particular within the DB community) one imposes safety conditions for Datalog rules.

Solution

- Unsafe negation would lead to infinite answer sets (if domain is infinite.)
- Variables occurring only in head would lead to domain dependance. For example, for $ans(x) \leftarrow R(a)$ all bindings for x in the domain of a DB where $R(a)$ is contained, would have to be in the set of answers. So the answer would not depend only on $R(a)$, i.e., only on the query, but also on the domain of the variables one allows.

Solution to Exercise 4.4 (4 points)

Give examples of general program rules for which

1. No fixed point exists at all (Hint: “This sentence is not true”)
2. Has two minimal fixed points (Hint: “The following sentence is false. The previous sentence is true.”)

Solution We consider propositional variables as 0-ary predicates. An extension of a propositional variable is then either the empty set $\emptyset$ which is interpreted as the truth value false, for short 0, or is the set consisting of the empty tuple $\{()\}$ which is interpreted as the truth value true, for short 1. Truthvalue assignments ν can be identified by the set of propositional variables which are assigned the value 1. So, e.g., $\nu(p) = 1, \nu(q)$ is represented by $\{p\}$, whereas $\nu(p) = 1, \nu(q) = 1$ is represented by $\{p, q\}$. So minimality on models becomes minimality w.r.t. set inclusion.

- No fixed point: $p \leftarrow \neg p$
- Two minimal fixed points.

$$q \leftarrow \neg p$$

$$p \leftarrow \neg q$$

Has minimal fixed points $\{p\}$ and $\{q\}$.