Reasoning about Schema Mappings *

Emanuel Sallinger

Vienna University of Technology
sallinger@dbai.tuwien.ac.at

Abstract

Schema mappings are an important tool in several areas of database research. Recently, the topic of *reasoning about schema mappings* was given attention, in particular revolving around the central concepts of *equivalence* and *optimality*. In this chapter, we survey these results. First, we introduce *relaxed notions of logical equivalence* and show their potential for finding optimized schema mappings. We then look at applications of these concepts to optimization, normalization, and schema mapping management, as well as the boundaries of computability. We conclude by giving a glimpse at reasoning about schema mappings in a broader sense by looking at how to debug schema mappings.

1998 ACM Subject Classification H.2.5 [Heterogeneous Databases]: Data translation

Keywords and phrases data exchange, data integration, schema mappings, equivalence, optimality, normalization, schema mapping management

Digital Object Identifier 10.4230/DFU.Vol5.10452.97

1 Introduction

Schema mappings are high-level specifications that describe the relationship between two database schemas. They are an important tool in several areas of database research, notably in data exchange [16, 8] and data integration [17, 15]. Over the past years, schema mappings have been extensively studied.

In this chapter, we will focus on the topic of reasoning about schema mappings. Central to any reasoning task is the concept of *implication*, and its close relative, *equivalence*. Since schema mappings are usually specified by logical formulas, the natural starting point of finding equivalence between schema mappings is

- **logical equivalence**: schema mappings that are satisfied by the same database instances are treated as being equivalent.

So now that we have a notion of equivalence, a natural next step is to use it to *optimize* schema mappings. That is, finding out the "best" among all equivalent schema mappings given some optimality criterion. Fortunately, there are algorithms for computing such optimized forms for a broad range of optimality criteria, as we will explore in Section 5.

Unfortunately, it turns out that logical equivalence is a quite restrictive concept for common tasks of data exchange. In particular, we can find two schema mappings where one is clearly preferred to the other, they both work perfectly well for the given data exchange task, but they are not logically equivalent. Hence we would not find the better one using optimization procedures for logical equivalence. This is clearly unsatisfactory.

* This work was supported by the Vienna Science and Technology Fund (WWTF), projects ICT08-032 and ICT12-15, and by the Austrian Science Fund (FWF): P25207-N23.

Data Exchange, Integration, and Streams. *Dagstuhl Follow-Ups*, Volume 5, ISBN 978-3-939897-61-3.
Editors: Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt; pp. 97–127

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

To remedy this situation, Fagin, Kolaitis, Nash and Popa in [9] introduced **relaxed notions of equivalence**: Notions that are less strict than logical equivalence, and therefore admit more optimization potential. These are

- **data-exchange equivalence**: schema mappings which behave in the same way for data-exchange are seen as equivalent, and
- **conjunctive-query equivalence**: schema mappings which behave similarly for answering conjunctive queries on the target database are treated as equivalent.

Therefore, using these notions, two of the prevalent applications of schema mappings can be reasoned about. The question remains of course: In which cases are there, hopefully efficient, algorithms for optimization under these relaxed notions of equivalence? This is a complex question that depends on the class of schema mappings we are interested in. We will talk about optimization potential and boundaries of computability in Section 6.

The notions of equivalence discussed up to now were primarily concerned with the two crucial tasks of data exchange and query answering. But beyond that, the area of **schema mapping management** [3, 4] poses quite different challenges: Operators of schema mapping management allow one to e.g. *invert* schema mappings or *extract* the essential parts of mappings.

The task of finding useful notions of equivalence between schema mappings for the purposes of schema mapping management was taken on by Arenas, Pérez, Reutter and Riveros in [1]. There they introduced notions of

- **equivalence in terms of information transfer**: schema mappings, which transfer the same amount of information are seen as equivalent. This notion has two variants, transferring source information and covering target information.

We will see that important operators of schema mapping management can be characterized using these equivalence notions and corresponding order relations in Section 7.

Up to now, we have talked about reasoning about mappings in a very strict sense. In the broad sense, *reasoning* about schema mappings covers a number of tasks related to working with schema mappings. When understanding and designing mappings, questions such as "What is this schema mapping doing?" and "Why is this schema mapping not doing what is expected?" are some of the first that are asked. We will give a glimpse at such reasoning tasks in a broader sense, like *analyzing* and *debugging* schema mappings, in Section 8.

1.1 Organization

In Section 2, we will introduce the necessary concepts. The main parts of this chapter are:

- **Concepts**: Where we will introduce notions of *equivalence* in Section 3 and then continue to discuss notions of *optimality* in Section 4.
- **Applications**: Where we will talk about optimization under logical equivalence in Section 5. After that, we look at the boundaries of computability in Section 6. We will finish by discussing applications to schema mapping management in Section 7.
- **Reasoning in the Broad Sense**: Where we will look at analyzing and debugging schema mappings in Section 8.

We finish this chapter with a conclusion and outlook in Section 9.

2 Preliminaries

In these preliminaries, we will first introduce database *schemas* and the relationships such as *homomorphisms* that may exist between database instances. Building upon that, we will define *schema mappings* and *solutions* to problems concerning schema mappings. After that, we describe the logical formalisms, called *dependencies*, on which schema mappings can be based. We conclude this section by introducing an algorithm called *the chase*.

2.1 Schemas

A **schema** $R = \{R_1, \ldots, R_n\}$ is a set of relation symbols R_i. Each relation symbol R_i has a fixed arity. An *instance* I over a schema R associates a relation R_i^I to each relation symbol in R. We call a relation symbol, together with some position an *attribute*. Sometimes, we associate a name to such an attribute. If $\vec{v} \in R_i^I$, we call $\vec{v}$ a *tuple* of R_i in I and also say that the *atom* $R_i(\vec{v})$ is contained in I. Instances in the context of this chapter are always considered to be finite. If the meaning is clear, we will not distinguish between syntax and semantics, e.g. relation symbols and relations. For two instances I, J, we write $I \subseteq J$ to say that the set of atoms contained in I is a subset of the set of atoms contained in J.

The **domain** of an instance I consists of two types of values, *constants* and *variables*. We write $\mathsf{dom}(I)$ for the domain, $\mathsf{const}(I)$ for the constants and $\mathsf{var}(I)$ for the variables. Variables are also called *labeled nulls* or *marked nulls*. We assume that $\mathsf{dom}(I) = \mathsf{var}(I) \cup \mathsf{const}(I)$ and $\mathsf{var}(I) \cap \mathsf{const}(I) = \emptyset$. An instance is called *ground*, if $\mathsf{var}(I) = \emptyset$. Instances are considered ground, unless specifically noted otherwise. In the same way as for instances, we can also refer to the domain, variables and constants of an atom, and speak of ground atoms. We usually denote labeled nulls by italic font (x) and constant symbols by sans-serif font (a).

Let $S = \{S_1, \ldots, S_n\}$ and $T = \{T_1, \ldots, T_m\}$ be schemas with no relation symbols in common. We write (S, T) to denote the combined schema $\{S_1, \ldots, S_n, T_1, \ldots, T_m\}$. If I is an instance of S and J is an instance of T, then (I, J) denotes the instance of the schema (S, T), consisting of the combined relations.

Let I, I' be instances. A **substitution** σ is a function $\mathsf{dom}(I) \to \mathsf{dom}(I')$ which replaces variables by constants or variables, but leaves constants unchanged, i.e., for all $c \in \mathsf{const}(I)$ it holds that $\sigma(c) = c$. We write $\sigma = [x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]$ if σ maps $x_i \in \mathsf{var}(I)$ to $a_i \in \mathsf{dom}(I)$ and for all $v \in \mathsf{dom}(I)$ not in $\{x_1, \ldots, x_n\}$, $\sigma(v) = v$.

A **homomorphism** $h\colon I \to I'$ is a substitution $\mathsf{dom}(I) \to \mathsf{dom}(I')$ (i.e. leaves constants unchanged) and for all atoms $R(\vec{x})$ it holds that $R(\vec{x}) \in I$ implies $R(h(\vec{x})) \in I'$. If there exists such an h, we write $I \to I'$. We say that I and I' are *homomorphically equivalent*, denoted $I \leftrightarrow J$, iff $I \to I'$ and $I' \to I$. If $I \to I'$ but not in the other direction, I is called *more general* than I', and I' is called *more specific* than I.

A homomorphism $h : I \to I'$ is called an *isomorphism*, iff h^{-1} is defined, and is a homomorphism from I' to I. If such an isomorphism exists, we write $I \cong I'$ and say that I and I' are *isomorphic*. A homomorphism $h : I \to I$ is called an *endomorphism*. An endomorphism is *proper* if it reduces the domain, that is, it is a surjective function (i.e. onto).

An instance $I^* \subseteq I$ is called a **core** of I , if $I \to I^*$ and I^* cannot be reduced by a proper endomorphism. That is, there is no $I' \subset I^*$ such that $I \to I'$. Cores have several important properties. The core is unique up to isomorphism, i.e. if I' and I'' are cores of I, then $I' \cong I''$. Therefore, we may talk about *the* core of I and refer to it as $\mathsf{core}(I)$. Furthermore, for instances I and I' it holds that $I \leftrightarrow I'$ iff $\mathsf{core}(I) \cong \mathsf{core}(I')$.

2.2 Schema mappings

We now introduce *schema mappings*, which specify the relationship between schemas. Based on the *data-exchange problem*, we will then explore specific instances called *solutions*, *universal solutions* and the *core of the universal solutions*. These are three of the key notions for working with schema mappings.

A **schema mapping** $\mathcal{M} = (S, T, \Sigma)$ is given by a *source schema* S, a *target schema* T, and a set Σ of dependencies over S and T in some logical formalism. An instance (I, J) of $\mathcal{M}$ is an instance of the schema (S, T), for which $(I, J) \models \Sigma$ holds. The instance I is called the *source instance* and is usually ground, whereas J is called the *target instance* and may contain variables. If the source schema S and the target schema T are clear, or the specific schema is not of primary interest, we will identify a schema mapping $\mathcal{M} = (S, T, \Sigma)$ with the set Σ of dependencies. We sometimes refer to schema mappings just as mappings.

A target instance J is called a **solution** *for* I *under* $\mathcal{M}$ if $(I, J) \models \Sigma$. The set of all solutions for I under $\mathcal{M}$ is denoted by $\mathsf{Sol}(I, \mathcal{M})$.

Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and I a source instance. A target instance J is a **universal solution** for I under $\mathcal{M}$ iff it is a solution for I under $\mathcal{M}$ and for all $J' \in \mathsf{Sol}(I, \mathcal{M})$ we have that $J \to J'$. The set of all universal solutions is denoted by $\mathsf{UnivSol}(I, \mathcal{M})$. An important property of universal solutions is that if J and J' are universal solutions for a source instance I under $\mathcal{M}$, they are homomorphically equivalent, i.e. $J \leftrightarrow J'$. Therefore the cores of J and J' are isomorphic, that is $\mathsf{core}(J) \cong \mathsf{core}(J')$ (cf. [10]).

The **core of the universal solutions** $\mathsf{core}(I, \mathcal{M})$ is given by $\mathsf{core}(I, \mathcal{M}) = \mathsf{core}(J)$ for any $J \in \mathsf{UnivSol}(I, \mathcal{M})$. Any universal solution J can be taken for computing the core, since the core of homomorphically equivalent instances is unique up to isomorphism. Note that the core of the universal solutions $\mathsf{core}(I, \mathcal{M})$ need not necessarily be a solution for I under $\mathcal{M}$.

2.3 Dependencies

Here we will focus on the logical formalisms called *dependencies* on which schema mappings are based. We will first define *embedded dependencies* and the important subtypes *tuple-generating* dependencies and *equality-generating* dependencies. These are all based on first-order logic. So after that, we will cover *second-order* dependencies. We conclude this section by a discussion of *conjunctive queries*.

An embedded **dependency** over a schema R is a first-order formula of the form

$$\forall \vec{x} \, (\varphi(\vec{x}) \to \exists \vec{y} \, \psi(\vec{x}, \vec{y}))$$

where φ is a conjunction of atoms over R called the *antecedent* and ψ is a conjunction of atoms over R and equalities called the *conclusion*. Furthermore, φ contains at least one atom, and each $x \in \vec{x}$ occurs at least once in φ.

The following notational convention will be adopted to save some space and ease reading. For dependencies, we will mostly omit universal quantifiers. All variables occurring in the antecedent are implicitly universally quantified. We will sometimes also omit existential quantifiers. All variables occurring just in conclusions are implicitly existentially quantified. Therefore, for the dependency $\forall \vec{x} \, (\varphi(\vec{x}) \to \exists \vec{y} \, \psi(\vec{x}, \vec{y}))$, we may write $\varphi(\vec{x}) \to \psi(\vec{x}, \vec{y})$.

A **tuple-generating dependency** (tgd) is an embedded dependency of the form

$$\forall \vec{x} \, (\varphi(\vec{x}) \to \exists \vec{y} \, \psi(\vec{x}, \vec{y}))$$

over a schema R where both φ and ψ are conjunctions of atoms over R. Tuple-generating dependencies can be viewed as generalizations of inclusion dependencies, in particular in the form of foreign-key constraints. However, tgds cannot express key constraints.

An **equality-generating dependency** (egd) is an embedded dependency of the form

$$\forall \vec{x}\,(\varphi(\vec{x}) \rightarrow x_i = x_j)$$

over a schema R where x_i and x_j are contained in $\vec{x}$. Equality-generating dependencies generalize functional dependencies. These are in particular used to express key constraints.

For a schema mapping $\mathcal{M} = (S, T, \Sigma)$, an embedded dependency $\phi(\vec{x}) \rightarrow \psi(\vec{x}, \vec{y})$ is called a **source-to-target dependency** (s-t dependency) if φ is defined over S and ψ over T. It is called a **target dependency** if both φ and ψ are defined over the T and called a *source dependency* if both are defined over S.

A **second-order tgd** (SO tgd) over source schema S and target schema T has the form

$$\exists \vec{f}\,(\tau_1 \wedge \ldots \wedge \tau_n) \qquad \text{where each } \tau_i \text{ has the form} \qquad \forall \vec{x}\,(\varphi(\vec{x}) \wedge \chi(\vec{x}) \rightarrow \psi(\vec{x}))$$

in which φ is conjunctions of atoms over S, ψ is a conjunction of atoms over T and χ is a conjunction of equalities. As values, atoms and equalities may contain function terms based on $\vec{f}$. That is, second-order tgds extend the notion of (first-order) s-t tgds by allowing existential quantification over function symbols. All variables from each $\vec{x}$ have to be safe. A variable is *safe*, if it occurs in the relational atoms of φ_i or is derived through equations or function applications from safe variables. Formal details can be found in [11].

By definition, it is clear that SO tgds are closed under conjunctions. A set of SO tgds can therefore be identified with a single SO tgd. The most important property of SO tgds is that they are also closed under composition [11].

Conjunctive queries. We conclude this section about logical formalisms by introducing *conjunctive queries*. They are important for one of the relaxed notions of equivalence that we are going to introduce.

A (Boolean) **conjunctive query** q over a schema R is a logical formula that has the form $\exists \vec{x}\,(A_1 \wedge \ldots \wedge A_n)$ where each A_i is an atom over relation symbols from R, and all variables occurring in q are from $\vec{x}$.

The **certain answers** to a (Boolean) conjunctive query q over T on a source instance I under a schema mapping $\mathcal{M}$, assuming that $\mathsf{Sol}(I, \mathcal{M}) \neq \emptyset$ are given as follows:

$$\mathsf{cert}(q, I, \mathcal{M}) = \bigcap_{J \in \mathsf{Sol}(I, \mathcal{M})} q(J)$$

The certain answers to a (Boolean) conjunctive query q on I under $\mathcal{M}$ can be obtained directly from a universal solution [8], that is $J \in \mathsf{UnivSol}(I, \mathcal{M})$ implies $\mathsf{cert}(q, I, \mathcal{M}) = \mathsf{ground}(q(J))$ where $\mathsf{ground}(q(j))$ denotes the ground atoms of $q(j)$, i.e., the atoms not containing variables. This concept can be naturally generalized to non-Boolean conjunctive queries.

2.4 The chase

The *chase* procedure [2] is an algorithm that computes universal solutions for a variety of schema mappings based on different dependency formalisms [8]. For a schema mapping $\mathcal{M} = (S, T, \Sigma)$ based on (finite sets of) s-t tgds, target tgds and egds, and SO tgds, the following holds: Given a source instance I, the chase procedure returns a universal solution J for I under $\mathcal{M}$, if it terminates and a universal solution exists. This J is called the **canonical universal solution** for I under $\mathcal{M}$ and written as $\mathsf{chase}(I, \mathcal{M})$.

The chase procedure computes a universal solution by a series of *chase steps*, based on a source instance and an initially empty target instance. In every step, a single dependency or multiple dependencies that are violated (i.e. not satisfied) by the current source and target instance are *applied* by adding further tuples to the target instance to fulfill those dependencies. We then say that these dependencies *fire*.

In the presence of target tgds, the chase does not always terminate. A sufficient condition for termination is that the set of target tgds is *weakly acyclic*. Intuitively, this criterion describes that the target tgds may not enter cycles which create new labeled nulls at each pass through the cycle. We refer to [19] and [8] for detailed definitions and further pointers.

We will use three variants of the chase in this chapter. They are in general based on s-t tgds, target tgds and target egds:

- the *standard* chase: in which for each step, one dependency that is violated is applied
- the *parallel* chase: in which for each step, all dependencies that are violated are applied
- the *SO tgd* chase: which is the chase procedure for SO tgds

More details and formal definitions of different variants of the chase procedure can be found in Chapter 1 of this book dedicated to the chase procedure.

2.5 Summary

This section introduced the most important preliminaries for the remainder of this chapter. A brief summary can be found in Figure 1.

A **schema mapping** $\mathcal{M} = (S, T, \Sigma)$ is given by

- a source schema S
- a target schema T
- a set Σ of dependencies over S and T in some logical formalism

Important dependency formalisms are

- **tuple-generating dependencies** (tgds)
  $\forall \vec{x} \, (\varphi(\vec{x}) \rightarrow \exists \vec{y} \, \psi(\vec{x}, \vec{y}))$
- **equality-generating dependencies** (egds)
  $\forall \vec{x} \, (\varphi(\vec{x}) \rightarrow x_i = x_j)$
- **second-order tgds** (SO tgds)
  $\exists \vec{f} \, (\tau_1 \wedge \ldots \wedge \tau_n)$ where each τ_i has the form
  $\forall \vec{x} \, (\varphi(\vec{x}) \wedge \chi(\vec{x}) \rightarrow \psi(\vec{x}))$ and χ is a conjunction of equalities

Figure 1 Schema mappings and dependencies (short summary).

3 Equivalence

In this first part on *concepts*, we will introduce the central notions of *equivalence* (this section) and *optimality* (next section) for schema mappings. We therefore start by covering the fundamental notions of equivalence we can use to compare schema mappings. We first treat

- **logical equivalence** between schema mappings, which equates schema mappings that are satisfied by the same instances.

Building upon that, we look at *relaxed notions of equivalence* introduced by Fagin et al. [9]. These notions characterize schema mappings that are not necessarily logically equivalent, but nevertheless indistinguishable for a variety of purposes. The first such relaxation is

- **data-exchange equivalence** (DE-equivalence), which equates schema mappings that exhibit the same behavior for data exchange. After that, we discuss
- **conjunctive-query equivalence** (CQ-equivalence), which equates schema mappings that yield the same certain answers to conjunctive queries.

If we need to compare schema mappings which differ e.g. in their target schemas, the amount of source information transferred becomes important, independently of how exactly this information is represented in the target instance [1]. This gives rise to

- **equivalence w.r.t. source information transferred** (S-equivalence). The corresponding notion for differing source schemas is
- **equivalence w.r.t. target information covered** (T-equivalence) based on the amount of target information that can be reconstructed by the schema mappings.

3.1 Notions of equivalence

We will now motivate and define these notions of equivalence. We will always start with an example, seeing why the respective notion naturally arises, and after that formally define that notion. Let us start by looking at such an example.

▶ **Example 1.** Over the source schema $S = \{P\}$ and target schema $T = \{Q\}$, let the schema mapping $\mathcal{M} = (S, T, \Sigma)$ be given by the following dependency:

- $P(x, y) \wedge P(z, y) \rightarrow Q(x, y)$

This tgd is very similar to a simple copy tgd from relation P to Q. However, the additional conjunct $P(z, y)$ in the antecedent seems superfluous. Indeed, since the variable z does not occur anywhere else in the dependency, and $P(z, y)$ is satisfied whenever $P(x, y)$ is satisfied, it is easy to see that the conjunct could just be left out.

To be more precise, the schema mapping $\mathcal{M}$ and the schema mapping $\mathcal{M}' = (S, T, \Sigma')$ given by the dependency

- $P(x, y) \rightarrow Q(x, y)$

are satisfied by exactly the **same pairs of instances**. ◀

In the previous example, we saw *logical equivalence* at work. This is the most natural notion to start with for reasoning about schema mappings, since our schema mappings are based on dependencies given in a logical formalism.

▶ **Definition 2.** Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T, \Sigma')$ be two schema mappings. $\mathcal{M}$ and $\mathcal{M}'$ are *logically equivalent* if for every source instance I and every target instance J,

$$(I, J) \models \Sigma \Leftrightarrow (I, J) \models \Sigma'$$

We denote logical equivalence by $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$. ◀

To avoid confusion, we do not use $\equiv$ without subscript in this chapter. The following formalization in terms of solutions for I under $\mathcal{M}$ characterizes the same notion. Two schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are *logically equivalent*, if for every source instance I, it holds that

$$\mathsf{Sol}(I, \mathcal{M}) = \mathsf{Sol}(I, \mathcal{M}')$$

This characterization follows immediately from the definition of solutions. The use of data exchange terminology for describing logical equivalence will be beneficial as we go on.

▶ **Example 3.** Over the source schema $S = \{P\}$ and target schema $T = \{Q, R\}$, let the schema mapping $\mathcal{M} = (S, T, \Sigma)$ be given by the following dependencies:

- $P(x, y) \to Q(x, y)$
- $R(x, y) \to R(x, x)$

Let us look at the result of data exchange under this schema mapping. We consider the source instance $I = \{P(\mathsf{a}, \mathsf{b})\}$ and compute the chase result $J = \mathsf{chase}(I, \mathcal{M}) = \{Q(\mathsf{a}, \mathsf{b})\}$. During this chase, the second dependency never fires. Even more striking, there is no universal solution for any I under $\mathcal{M}$ which ever materializes a tuple of R.

So, for the purposes of data exchange which is usually concerned with universal solutions, we would like to simplify $\mathcal{M}$ into the schema mapping $\mathcal{M}' = (S, T, \Sigma')$ given by the dependency

- $P(x, y) \to Q(x, y)$

Unfortunately, $\mathcal{M}$ and $\mathcal{M}'$ are not logically equivalent, they do not have the same solutions for every source instance. Consider, for the source instance $I = \{P(\mathsf{a}, \mathsf{b})\}$, the solution $J = \{Q(\mathsf{a}, \mathsf{b}), R(\mathsf{a}, \mathsf{b})\}$. This clearly violates $\mathcal{M}$, since the tuple $R(\mathsf{a}, \mathsf{a})$ would be required by the second dependency.

But, as we have seen before, for the purposes of data exchange, the schema mappings are "just as good". To be more precise, the schema mappings $\mathcal{M}$ and $\mathcal{M}'$ have the **same universal solutions** for all source instances. ◀

The previous example motivates the introduction of the first relaxed notion of equivalence introduced by Fagin et al. [9], *data-exchange equivalence* (DE-equivalence), which does not distinguish schema mappings which behave in the same way for the purposes of data exchange, or more formally:

▶ **Definition 4.** [9] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T, \Sigma')$ be two mappings. $\mathcal{M}$ and $\mathcal{M}'$ are *data-exchange equivalent*, if for every source instance I,

$$\mathsf{UnivSol}(I, \mathcal{M}) = \mathsf{UnivSol}(I, \mathcal{M}')$$

We denote data-exchange equivalence by $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$. ◀

Since for logical equivalence *all solutions* coincide, and for data-exchange equivalence the *universal solutions* coincide, it is clear that from $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$, it follows that $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$ for all schema mappings. With that, it is appropriate to talk about a *relaxation* of logical equivalence. Also, in Example 3, we have already seen that this relaxation is proper (i.e. both notions are distinct) for schema mappings based on s-t tgds and target tgds.

▶ **Example 5.** Over the source schema $S = \{P\}$ and target schema $T = \{Q\}$, let the schema mapping $\mathcal{M} = (S, T, \Sigma)$ be given by the following dependencies:

- $P(x, y) \to Q(x, x)$
- $Q(x, y) \to Q(x, x)$

Now, compared to the previous example, the source-to-target dependency is not anymore a simple copy tgd, and the relation symbol occurring in the target dependency actually also occurs in the source-to-target one. Can we, as in the previous example, simply remove the second dependency, gaining the schema mapping $\mathcal{M}' = (S, T, \Sigma')$ given by

- $P(x, y) \to Q(x, y)$

while still upholding data-exchange equivalence? It is now a bit more subtle to see why $\mathcal{M}$ and $\mathcal{M}'$ do not have the same universal solutions. The more so as for the source

instance $I = \{P(\mathsf{a}, \mathsf{b})\}$, both schema mappings have the same canonical universal solution $J = \mathsf{chase}(I, \mathcal{M}) = \mathsf{chase}(I, \mathcal{M}') = \{Q(\mathsf{a}, \mathsf{a})\}$.

But now consider $J' = \{Q(\mathsf{a}, \mathsf{a}), Q(u, v)\}$ where u and v are variables. It is clear that J' is still a universal solution under $\mathcal{M}'$, since it is a solution under $\mathcal{M}'$, and J' maps to the universal solution J through the homomorphism $[u \mapsto \mathsf{a}, v \mapsto \mathsf{a}]$. Yet to satisfy $\mathcal{M}$, since $u \neq v$, it would require the atom $Q(u, u)$ to be present. So J' is not a universal solution for I under $\mathcal{M}$.

This state of affairs is clearly unsatisfactory if we look at a conjunctive query like $\exists x, y\, Q(x, y)$. If we want the certain answer to this query, a "strange" universal solution like J' will not affect the result. The tuple $Q(u, v)$ from J' will not be contained in the certain answers, since it is not contained in J. Indeed, $\mathcal{M}$ and $\mathcal{M}'$ will yield the **same certain answers to conjunctive queries**. ◀

This example directly leads us to *conjunctive-query equivalence* (CQ-equivalence). It is based on the behavior of conjunctive queries, posed against the solutions of a schema mapping.

▶ **Definition 6.** [9] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T, \Sigma')$ be two schema mappings. $\mathcal{M}$ and $\mathcal{M}'$ are *conjunctive-query equivalent*, if for every source instance I and every conjunctive query q, either $\mathsf{Sol}(I, \mathcal{M}) = \mathsf{Sol}(I, \mathcal{M}') = \emptyset$ or

$$\mathsf{cert}(q, I, \mathcal{M}) = \mathsf{cert}(q, I, \mathcal{M}')$$

We denote conjunctive-query equivalence by $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$. ◀

By this definition, there can be two reasons for schema mappings to be CQ-equivalent. The first is that both schema mappings could have no solutions at all. In this case, we say that the schema mappings are CQ-equivalent. The other case is of course that there are solutions to both schema mappings, and the certain answers to queries against them coincide.

The original definition of CQ-equivalence given above is based on the certain answers to conjunctive queries. However, an alternative characterization is possible:

▶ **Proposition 7.** *[9] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T, \Sigma')$ be two schema mappings such that the following holds.*

$$\mathsf{Sol}(I, \mathcal{M}) \neq \emptyset \quad \textit{implies} \quad \mathsf{UnivSol}(I, \mathcal{M}) \neq \emptyset$$

$\mathcal{M}$ and $\mathcal{M}'$ are conjunctive-query equivalent*, if for every source instance I, either $\mathsf{Sol}(I, \mathcal{M}) = \mathsf{Sol}(I, \mathcal{M}') = \emptyset$ or*

$$\mathsf{core}(I, \mathcal{M}) = \mathsf{core}(I, \mathcal{M}')$$ ◀

A few points are of interest now. The first crucial question is for which classes of schema mappings it holds that $\mathsf{Sol}(I, \mathcal{M}) \neq \emptyset$ implies $\mathsf{UnivSol}(I, \mathcal{M}) \neq \emptyset$. That is, for which kinds of schema mappings is there always a universal solution whenever any solution exists. In [9], it is shown that a sufficient condition is to have schema mappings defined by s-t tgds, target egds, target tgds that have a terminating chase, as well as to SO tgds. In particular, leaving out target tgds or requiring the set of target tgds to be weakly acyclic guarantees a terminating chase and therefore the property that we require.

What is also clear now is that in this case, CQ-equivalence is in fact a relaxation of DE-equivalence and therefore of logical equivalence. This was not obvious for the characterization based on conjunctive queries. It is easy to see here though, since the core is based on some universal solution and by DE-equivalence all universal solutions coincide, so from $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$ follows $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$.

We have now arrived at a *hierarchy of schema-mapping equivalences* given by respective relaxation between logical equivalence, DE-equivalence and CQ-equivalence. This hierarchy itself holds for *all* classes of schema mappings. For the most important classes of schema mappings (where universal solutions exist, given that solutions exist), one can easily see this: given that *all* solutions coincide, the *universal* solutions coincide and from that follows that the *cores* of the universal solutions coincide.

▶ **Proposition 8.** *[9] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T, \Sigma')$ be two schema mappings*

$$\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}' \quad \Rightarrow \quad \mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}' \quad \Rightarrow \quad \mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$$

◀

Also, we have seen in Examples 3 and 5 that this hierarchy is proper for schema mappings based on s-t tgds and target tgds. Later in this chapter, we will take a more detailed look at for which classes of schema mappings the hierarchy is proper, and for which it collapses.

We now have at hand two very natural relaxations of logical equivalence. Next we will explore what happens if we need to reason about schema mappings that have differing source or target schemas. They were introduced by Arenas et al. in [1].

▶ **Example 9.** Over the source schema $S = \{P\}$ and target schema $T = \{Q\}$, let the schema mapping $\mathcal{M} = (S, T, \Sigma)$ be given by the following dependencies:

- $P(x, y) \to Q(x, y)$

Now consider the slightly altered schema mapping $\mathcal{M}' = (S, T', \Sigma')$ for $T' = \{R\}$ given by

- $P(x, y) \to R(x, y, y)$

Since the target relations affected by the two schema mappings are different, it is clear that $\mathcal{M}$ and $\mathcal{M}'$ are neither logically, nor DE-, nor CQ-equivalent. But intuitively, these schema mappings are very similar.

In fact, through a schema mapping $\mathcal{N}$ based on $Q(x, y) \to R(x, y, y)$ we can "reconstruct" $\mathcal{M}'$ from $\mathcal{M}$ in the following sense: The composition of $\mathcal{M}$ and $\mathcal{N}$ yields $\mathcal{M}'$. In the same way, we can reconstruct $\mathcal{M}$ through the composition of $\mathcal{M}'$ and a schema mapping $\mathcal{N}'$ based on $R(x, y, y) \to Q(x, y)$.
In this way, we see that both schema mappings **transfer the same amount of source information**, in the sense that they are able to reconstruct the result of the other. ◀

▶ **Definition 10.** [1] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T', \Sigma')$ be two schema mappings. $\mathcal{M}'$ *transfers at least as much source information as* $\mathcal{M}$, written $\mathcal{M} \preceq_{\mathsf{S}} \mathcal{M}'$, iff there exists a schema mapping $\mathcal{N}$ from T to T' s.t.

$$\mathcal{M} \circ \mathcal{N} \equiv_{\mathsf{log}} \mathcal{M}'$$

We say that $\mathcal{M}$ and $\mathcal{M}'$ are *equivalent w.r.t. the source information transferred*, written $\mathcal{M} \equiv_{\mathsf{S}} \mathcal{M}'$, iff $\mathcal{M} \preceq_{\mathsf{S}} \mathcal{M}'$ and $\mathcal{M}' \preceq_{\mathsf{S}} \mathcal{M}$. ◀

Given that we talked about schema mappings with differing target schemas, it is natural to ask about differing source schemas. The following definition mirrors the above one:

▶ **Definition 11.** [1] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S', T, \Sigma')$ be two schema mappings. $\mathcal{M}'$ *covers at least as much target information as* $\mathcal{M}$, written $\mathcal{M} \preceq_{\mathsf{T}} \mathcal{M}'$, iff there exists a schema mapping $\mathcal{N}$ from S to S' s.t.

$$\mathcal{N} \circ \mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$$

We say that $\mathcal{M}$ and $\mathcal{M}'$ are *equivalent w.r.t. the target information covered*, written $\mathcal{M} \equiv_{\mathsf{T}} \mathcal{M}'$, iff $\mathcal{M} \preceq_{\mathsf{T}} \mathcal{M}'$ and $\mathcal{M}' \preceq_{\mathsf{T}} \mathcal{M}$. ◀

Note that while the names of the ordering relations are given in this way in [1], the corresponding equivalence relations are originally used without reference to a specific name.

3.2 Summary

In this section, we introduced notions of equivalence for schema mappings and showed through examples how they naturally arise when working with schema mappings.

We started with *logical equivalence* and then introduced *relaxed notions of equivalence*: notions of equivalence which do not distinguish between mappings which behave the same for a given purpose. We discussed *data-exchange equivalence* and *conjunctive-query equivalence*. After that, we looked at two notions of *equivalence in terms of information transfer*.

In total, for schema mappings where the existence of solutions implies the existence of universal solutions, the definitions are for reference summarized in Figure 2.

Two schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are

- **logically equivalent** ($\equiv_{\mathsf{log}}$)
  iff for all I we have $\mathsf{Sol}(I, \mathcal{M}) = \mathsf{Sol}(I, \mathcal{M}')$
- **data-exchange equivalent** ($\equiv_{\mathsf{DE}}$)
  iff for all I we have $\mathsf{UnivSol}(I, \mathcal{M}) = \mathsf{UnivSol}(I, \mathcal{M}')$
- **conjunctive-query equivalent** ($\equiv_{\mathsf{CQ}}$)
  iff* for all I we have $\mathsf{core}(I, \mathcal{M}) = \mathsf{core}(I, \mathcal{M}')$
- **equivalent w.r.t. source information transferred** ($\equiv_{\mathsf{S}}$)
  iff there exist $\mathcal{N}, \mathcal{N}'$ s.t. $\mathcal{M} \circ \mathcal{N} \equiv_{\mathsf{log}} \mathcal{M}'$ and $\mathcal{M}' \circ \mathcal{N}' \equiv_{\mathsf{log}} \mathcal{M}$
- **equivalent w.r.t. target information covered** ($\equiv_{\mathsf{T}}$)
  iff there exist $\mathcal{N}, \mathcal{N}'$ s.t. $\mathcal{N} \circ \mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$ and $\mathcal{N}' \circ \mathcal{M}' \equiv_{\mathsf{log}} \mathcal{M}$

Figure 2 Notions of equivalence (*assuming universal solutions exist in case solutions exist).

4 Optimality

Given that we can reason about schema mappings that are equivalent according to a variety of notions as introduced in the previous section, there is a natural next task at hand: Finding a schema mapping that is "best" among those equivalent mappings.

In this section, we therefore discuss a number of optimality criteria. Mostly, we will talk about notions of "minimality" or "redundancy". These give rise to decision problems, i.e. identifying if a given schema mapping is minimal or non-redundant among equivalent schema mappings. They of course also induce optimization problems in the sense of actually finding such minimal or non-redundant schema mappings.

This section starts with the most basic optimality criteria, like subset- and cardinality-minimality. There, our main concern will be understanding how they affect schema mappings. But there are also quite intricate optimality criteria that we will talk about.

We thus start with some of the most basic optimality criteria (formal definitions follow later):

- **σ-redundancy**: By that we mean detecting if some specific dependency σ is redundant, i.e. could be left out while still yielding an equivalent schema mapping. Closely related is
- **subset-minimality**: That is, finding a minimal subset of dependencies. In other words, that set should contain no dependency that is redundant. A natural next step is
- **cardinality-minimality**: Finding a schema mapping that uses the minimum number of dependencies possible.

The previous three notions were concerned with schema mappings at the level of dependencies, but did not look inside of those dependencies. In [14], further criteria were presented that are concerned with internal characteristics of the given dependencies. There is, given in slightly generalized form:

- **antecedent-minimality**: It is concerned with the total number of atoms in antecedents. Together with cardinality-minimality, this aims at reducing the computational cost of the joins computed by the chase. The complementary notion is
- **conclusion-minimality**: Minimizing the total number of atoms in the conclusions. Besides the number of atoms, we can also consider
- **variable-minimality**: It is based on minimizing the total number of existentially quantified variables. This is of course related to the number of labeled nulls introduced during the chase.

The previously mentioned optimality criteria were all syntactically defined, which is important for the computational cost of the chase or similar procedures. Still, there are interesting semantic optimality criteria that are not based on the dependencies, but on the mapping seen as a binary relation between source and target instances. The following semantic criteria were introduced in [1] w.r.t. specific notions of equivalence:

- **target-redundancy**: Is there an equivalent schema mapping that "uses fewer target instances" (in the sense that the range of the optimized schema mapping is a subset of the range of the original one). The complementary notions is
- **source-redundancy**: Is there an equivalent schema mapping with a subset of source instances. We will give formal definitions of all notions later in this section.

For all of the criteria, two things need to be fixed: First, what is the *notion of equivalence* we are talking about? Secondly, what is the class of schema mappings that we allow for the desired optimized mapping? The interplay between notions of optimality, notions of equivalence and desired classes of schema mappings will turn out to be interesting.

4.1 Notions of optimality

We will now motivate and define these notions of optimality. Like in the previous section, we will always start with an example, seeing why the respective notion naturally arises, and after that formally define that notion. Let us start by looking at such an example.

▶ **Example 12.** Consider the schema mapping $\mathcal{M}$ given by the following dependencies:

- $P(x, y) \to Q(x, y)$ (σ_1)
- $P(x, x) \to Q(x, x)$ (σ_2)
- $P(x, y) \to R(y)$ (σ_3)
- $P(u, v) \to R(v)$ (σ_4)

Looking at dependency σ_2, it is clear that whenever a tuple is produced through σ_2, the same tuple is also produced by σ_1. That is, σ_2 is clearly redundant in $\mathcal{M}$ (w.r.t. logical equivalence). ◀

▶ **Definition 13.** Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\sigma \in \Sigma$. We say that $\mathcal{M}$ *is σ-redundant w.r.t. e-equivalence*, iff $\Sigma \setminus \{\sigma\} \equiv_e \Sigma$. ◀

▶ **Example 12 (ctd).** We have seen that $\mathcal{M}$ is σ_2-redundant, and we can remove it while retaining logical equivalence. It is also easy to see that $\mathcal{M}$ is both σ_3 and σ_4-redundant, since they are isomorphic "copies" of each other. Still, simply removing all σ-redundant

dependencies will not yield a logically equivalent schema mapping: either σ_3 or σ_4 needs to be retained. Thus, the following schema mapping $\mathcal{M}'$ given by Σ' as

- $P(x,y) \to Q(x,y)$ (σ_1)
- $P(x,y) \to R(y)$ (σ_3)

is a minimal subset of Σ s.t. $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$. ◀

▶ **Definition 14.** Let $\mathcal{M} = (S,T,\Sigma)$ be a schema mapping and $\sigma \in \Sigma$. We say that $\mathcal{M}$ *is subset-minimal w.r.t. e-equivalence*, iff there is no schema mapping $\mathcal{M}' = (S,T,\Sigma')$ s.t. $\Sigma' \subset \Sigma$ and $\mathcal{M} \equiv_e \mathcal{M}'$. ◀

▶ **Example 12 (ctd).** Somehow, the schema mapping $\mathcal{M}'$ is still not completely satisfactory regarding the number of dependencies. If we talk about *subsets*, $\mathcal{M}'$ is clearly the best we can do, but if we allow *arbitrary* dependencies, we can define $\mathcal{M}''$ based on

- $P(x,y) \to Q(x,y) \wedge R(y)$ (σ_5)

This clearly has the minimum cardinality among all logically equivalent schema mappings. ◀

In the previous example, we have seen that $\mathcal{M}''$, is cardinality minimal among schema mappings based on arbitrary dependencies. If we only look at schema mappings based on GAV dependencies (which restrict tgds by allowing only a single atom in the conclusion), we see that $\mathcal{M}'$ is cardinality minimal. This motivates the following definition relative to the class of schema mappings:

▶ **Definition 15.** Let $\mathcal{M} = (S,T,\Sigma)$ be a schema mapping and $\mathcal{C}$ a class of schema mappings. We say that $\mathcal{M}$ *is cardinality-minimal w.r.t. e-equivalence among $\mathcal{C}$-schema mappings*, iff there is no mapping $\mathcal{M}' = (S',T',\Sigma')$ in $\mathcal{C}$ s.t. $|\Sigma'| < |\Sigma|$ and $\mathcal{M} \equiv_e \mathcal{M}'$. ◀

Up to now, we have looked only at the dependencies themselves. We will now look inside of them to find additional ways to optimize these schema mappings:

▶ **Example 16.** Consider the schema mapping $\mathcal{M}$ given by the following dependency:

- $P(x,y) \wedge P(u,v) \to Q(x,y)$

It is clear that we could just as well leave out the atom $P(u,v)$ in the antecedent, thus getting the schema mapping $\mathcal{M}'$ based on

- $P(x,y) \to Q(x,y)$

which has the minimum total number of atoms in the antecedent (based on all schema mappings that are logically equivalent). ◀

This motivates the following definition. Note that we are talking about the *total* number of atoms over all dependencies here, not the *maximum* number over all dependencies.

▶ **Definition 17.** [14] Let $\mathcal{M} = (S,T,\Sigma)$ be a schema mapping and $\mathcal{C}$ a class of sch. mappings. We say that $\mathcal{M}$ *is antecedent-minimal w.r.t. e-equivalence among $\mathcal{C}$-schema mappings*, iff there is no mapping $\mathcal{M}' = (S',T',\Sigma')$ in $\mathcal{C}$ s.t. $\mathsf{AntSize}(\Sigma') < \mathsf{AntSize}(\Sigma)$ and $\mathcal{M} \equiv_e \mathcal{M}'$, where $\mathsf{AntSize}$ denotes the total number of atoms in antecedents. ◀

▶ **Example 18.** Consider the schema mapping $\mathcal{M}$ given by the following dependency:

- $P(x,y) \to \exists z(Q(x,y) \wedge Q(x,z))$

There are two dimensions in the conclusion that we can measure: the number of *atoms*, and the number of *existential variables* that we use. The following mapping $\mathcal{M}'$

- $P(x,y) \to Q(x,y)$

uses both the minimum total number of atoms in the conclusion, as well as the minimum total number of existentially quantified variables. ◀

▶ **Definition 19.** [14] Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\mathcal{C}$ a class of sch. mappings:

- $\mathcal{M}$ *is conclusion-minimal w.r.t. e-equivalence among* $\mathcal{C}$*-schema mappings*, iff there is no schema mapping $\mathcal{M}' = (S', T', \Sigma')$ in $\mathcal{C}$ s.t. $\mathsf{ConSize}(\Sigma') < \mathsf{ConSize}(\Sigma)$ and $\mathcal{M} \equiv_e \mathcal{M}'$, where $\mathsf{ConSize}$ denotes the total number of atoms in conclusions.
- $\mathcal{M}$ *is variable-minimal w.r.t. e-equivalence among* $\mathcal{C}$*-schema mappings*, iff there is no schema mapping $\mathcal{M}' = (S', T', \Sigma')$ in $\mathcal{C}$ s.t. $\mathsf{VarSize}(\Sigma') < \mathsf{VarSize}(\Sigma)$ and $\mathcal{M} \equiv_e \mathcal{M}'$, where $\mathsf{VarSize}$ denotes the total number of existentially quantified variables. ◀

We can now talk about schema mappings based on their dependencies opaquely, and we can look inside of those dependencies based on the number of atoms and existentially quantified variables. For many tasks in data exchange and data integration, above all for the chase procedure, it can be argued that it is desirable to find schema mappings which are minimal under some, if not all of those criteria.

Still there is another, semantical, point of view in which such a schema mapping can still be redundant, and it will have important applications later on:

▶ **Example 20.** Consider the schema mapping $\mathcal{M}$ given by the dependency:

- $P(x, y) \to Q(x, x)$

Intuitively, this schema mapping "wastes space" compared to a schema mapping based on e.g. $P(x, y) \to R(x)$ by storing each source value x twice in the target.

In a more precise way, $\mathcal{M}$ is redundant in the following sense: Given source instance $I = \{P(\mathsf{a}, \mathsf{b})\}$, the canonical universal solution is $J = \{Q(\mathsf{a}, \mathsf{a}))\}$. But there is also another possible solution $J' = \{Q(\mathsf{a}, \mathsf{a})), Q(\mathsf{a}, \mathsf{b}))\}$ with $J \subset J'$. And indeed, we can find another schema mapping $\mathcal{M}'$ which has J as a solution, but not J':

- $P(x, y) \to Q(x, x)$
- $Q(x, y) \to x = y$

Clearly $\mathcal{M} \equiv_{\mathsf{S}} \mathcal{M}'$, that is, they transfer the same amount of source information (incidentally, they are also CQ-equivalent). In total, we have two mappings $\mathcal{M}$ and $\mathcal{M}'$, both are equivalent w.r.t. source information transferred, but one has a strict subset of solutions for I. ◀

▶ **Definition 21.** Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\mathcal{C}$ a class of schema mappings. $\mathcal{M}$ is *target-redundant w.r.t. e-equivalence among* $\mathcal{C}$ *schema mappings*, iff there is a target instance $J' \in \{J \mid (I, J) \in \mathcal{M}\}$ s.t. for $\mathcal{M}' = \{(I, J) \in \mathcal{M} \mid J \neq J'\}$ holds $\mathcal{M} \equiv_e \mathcal{M}'$. ◀

The preceding definition was originally given in [1] w.r.t. S-equivalence. As we will later see, this is also the way it is commonly used and if no other notion of equivalence is explicitly mentioned, we assume this notion as default.

As an important remark, note that this definition does not necessarily talk about schema mappings "wasting space" *inside* the target instances. In particular, both schema mappings $\mathcal{M}$ and $\mathcal{M}'$ in the previous example store the value x twice in the target. So in this sense, they both waste space, even though one is target redundant and the other is not (w.r.t. S-equivalence among all schema mappings). The point is that $\mathcal{M}$ wastes target *instances*, since a subset of those would suffice.

Let us conclude this section by defining the natural counterpart to target-redundancy: source-redundancy. It is commonly used w.r.t. T-equivalence.

▶ **Definition 22.** Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\mathcal{C}$ a class of schema mappings. $\mathcal{M}$ is *source-redundant w.r.t. e-equivalence among* $\mathcal{C}$ *schema mappings*, iff there is a source instance $I' \in \{I \mid (I, J) \in \mathcal{M}\}$ s.t. for $\mathcal{M}' = \{(I, J) \in \mathcal{M} \mid I \neq I'\}$ holds $\mathcal{M} \equiv_e \mathcal{M}'$. ◀

4.2 Summary

We now have means to talk about non-redundant and minimal schema mappings in a variety of manners: We can have schema mappings that are non-redundant in their syntactic representation (dependencies, atoms, variables) and their semantic extension (source- and target instances). For reference, we informally summarize the definitions in Figure 3.

The following criteria are given w.r.t. a notion of equivalence and a class of mappings:

- σ-**redundant**: a *specific* dependency σ could be left out
- **subset-minimal**: no dependency could be left out
- **cardinality-minimal**: the number of dependencies is minimal

- **antecedent-minimal**: the *total* number of atoms in antecedents is minimal
- **conclusion-minimal**: the *total* number of atoms in conclusions is minimal
- **variable-minimal** the *total* number of *existentially* quantified variables is minimal

- **target-redundant**: a target instance could be left out
- **source-redundant**: a source instance could be left out

All criteria are usually w.r.t. logical equivalence, the exceptions are that usually target-redundancy is w.r.t. S-equivalence, source-redundancy w.r.t. T-equivalence.

Figure 3 Notions of optimality for a schema mapping (informal summary).

5 Normalization and optimization for logical equivalence

In the previous sections, our main goal was to develop the relevant notions of equivalence and optimality for reasoning about schema mappings. What was left open was *how* to use these notions for actual reasoning, that is, the question of algorithms and complexity.

In this section, we will talk about reasoning under logical equivalence. The major result we will cover here, presented by Gottlob et al. in [14], is that there is an algorithm which transforms schema mappings based on s-t tgds into an optimal form in the following sense:

- it is a *unique normal form* (up to variable renaming), and
- the mapping is cardinality-, antecedent-, conclusion- and variable-minimal among all *split-reduced* schema mappings.

This form can be computed in polynomial time if the length of each dependency is bounded by a constant. What exactly split-reduced schema mappings are will be our next topic:

5.1 Finding optimal split-reduced schema mappings

Let us see why optimality among *all* mappings based on s-t tgds is not always desirable:

▶ **Example 23.** Over the source schema $S = \{L\}$ and the target schema $T = \{C, E\}$, let the schema mapping $\mathcal{M}$ be given by the following dependencies:

- $L(x_1, x_2, x_3) \rightarrow \exists y\, C(x_1, y)$ (σ_1)
- $L(x_1, x_2, x_3) \wedge L(x_4, x_2, x_5) \rightarrow E(x_1, x_4)$ (σ_2)

It is easy to see that the antecedents of σ_1 is fulfilled whenever the antecedent of σ_2 is fulfilled. Therefore, clearly $\mathcal{M}$ is logically equivalent to $\mathcal{M}'$ based on the following dependency

- $L(x_1, x_2, x_3) \wedge L(x_4, x_2, x_5) \rightarrow \exists y(C(x_1, y) \wedge E(x_1, x_4))$ (σ_3)

which contains both conclusions in a single dependency.

Now let us compute the canonical universal solutions to both of these schema mappings for source instance $I = \{L(\mathsf{a}, \mathsf{b}, \mathsf{c}), L(\mathsf{d}, \mathsf{b}, \mathsf{f})\}$ (which is just the antecedent of σ_3 with variables replaced by distinct constants). Let $J = \mathsf{chase}(I, \mathcal{M})$ and $J' = \mathsf{chase}(I, \mathcal{M}')$.

Considering the C-atoms, for J we have $C(\mathsf{a}, y_1)$ and $C(\mathsf{d}, y_2)$, since there are two possible ways to instantiate the antecedent of σ_1. But for J', since for σ_3 there are actually four possible ways to instantiate the antecedent, we additionally have $C(\mathsf{a}, y_3)$ and $C(\mathsf{d}, y_4)$. ◀

In the preceding example, we saw that, while $\mathcal{M}'$ is the cardinality-, antecedent- and conclusion-minimal mapping among *all* schema mappings, this leads to a quadratic blowup of the size of the canonical universal solution compared to $\mathcal{M}$.

However, σ_3 of $\mathcal{M}'$ has a problematic property: The atoms in the conclusion are actually not related to each other, they could very well be formulated in separate dependencies. This was the reason for the quadratic blowup. Therefore the following was defined:

▶ **Definition 24.** [14] A schema mapping $\mathcal{M} = (S, T, \Sigma)$ consisting of s-t tgds is *split-reduced*, if there is no logically equivalent mapping $\mathcal{M}' = (S, T, \Sigma')$ with $|\Sigma| > |\Sigma'|$ but $\mathsf{ConSize}(\Sigma) = \mathsf{ConSize}(\Sigma')$. ◀

Let us look at what this definition means: If we can, through "splitting up" a dependency – thus raising the number of dependencies – still have the same total size of the conclusions, then some conclusion atoms where not related to each other in a significant way. In other words, the dependencies are decomposed without raising the total size of the conclusions.
If we find a schema mapping among the class of *split-reduced* schema mappings that is minimal according to our chosen criteria, we get both a schema mapping that has good properties in terms of the dependencies (minimality) and good properties in the solution produced by the chase (some unnecessary blowup is avoided).

Note that one can view split-reducedness also as a derived optimality criterion like the ones discussed in the previous section (based on cardinality- and conclusion minimality). Let us now find, through an example, rules to rewrite a mapping into the optimal form we promised. For ease of reference, the rewrite rule numbers will match those in [14]:

▶ **Example 25** (based on [14])**.** Over the source schema $S = \{L\}$ and $T = \{P, Q, R\}$, let the mapping $\mathcal{M}$ be given by the following dependencies. For readability, all variables x_i are universally quantified and all variables y_i are existentially quantified. For the same reason, we use constant a within dependencies (avoidable by e.g. adding $A(\mathsf{a})$ to all antecedents).

- $L(x_1, x_2, x_3) \to P(x_1, y_1, \mathsf{a}) \wedge R(y_1, x_2, \mathsf{a}) \wedge R(y_1, x_2, y_2)$ (σ_1)
- $L(x_1, x_1, x_1) \to P(x_1, y_1, y_2) \wedge Q(y_2, y_3, x_1) \wedge R(y_1, x_1, y_2)$ (σ_2)
- $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \to P(x_1, y_2, y_1) \wedge Q(y_1, y_3, x_2) \wedge Q(\mathsf{a}, y_3, x_2) \wedge R(x_2, y_4, x_3)$ (σ_3)

In this state, it is very hard for a human to make much sense out of this schema mapping without significant analysis. Let us therefore try to simplify it before trying to understand it.

A simple first rewriting is for σ_1: The last atom $R(y_1, x_2, y_2)$ is actually a more specific form of the second one $R(y_1, x_2, \mathsf{a})$ in the conclusion. Thus we can do the following:

Rule 1: Simplify the conclusion to its core
applied to: $L(x_1, x_2, x_3) \to P(x_1, y_1, \mathsf{a}) \wedge R(y_1, x_2, \mathsf{a}) \wedge R(y_1, x_2, y_2)$ (σ_1)

So through the homomorphism $[y_2 \mapsto \mathsf{a}]$, we can drop the last atom arriving at the core of the conclusion. Before making things easier by dropping further atoms, let us try to split up the quite long dependency σ_3, to get a better overview (and reach a split-reduced form):

Rule 3: Split the dependency if possible
applied to: $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \rightarrow P(x_1, y_2, y_1) \wedge Q(y_1, y_3, x_2) \wedge Q(\mathsf{a}, y_3, x_2) \wedge R(x_2, y_4, x_3)$ (σ_3)

We see that in the first three atoms of the conclusion, there are existentially quantified variables y_1 to y_3 intermingled and in the last one, there is only y_4. Indeed, we can split σ_4 in this way, yielding two dependencies, one with the conclusion $R(x_2, y_4, x_3)$ and one with the other three atoms. Let us look at the current state of the schema mapping after having applied those two rules:

- $L(x_1, x_2, x_3) \rightarrow P(x_1, y_1, \mathsf{a}) \wedge R(y_1, x_2, \mathsf{a})$ (σ'_1)
- $L(x_1, x_1, x_1) \rightarrow P(x_1, y_1, y_2) \wedge Q(y_2, y_3, x_1) \wedge R(y_1, x_1, y_2)$ (σ_2)
- $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \rightarrow P(x_1, y_2, y_1) \wedge Q(y_1, y_3, x_2) \wedge Q(\mathsf{a}, y_3, x_2)$ (σ'_3)
- $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \rightarrow R(x_2, y_4, x_3)$ (σ'_4)

Now we look a bit further at the new rule σ'_3. Following the split, in the antecedent there occurs variable x_3, but it is never used in the conclusion. Therefore we can:

Rule 2: Simplify the antecedent to its core
applied to: $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \rightarrow P(x_1, y_2, y_1) \wedge Q(y_1, y_3, x_2) \wedge Q(\mathsf{a}, y_3, x_2)$ (σ'_3)

Easily, through the homomorphism $[x_3 \mapsto x_2]$, we can thereby drop the second atom of the antecedent.

Let us stay with the conclusion of this dependency. Take the first atom $P(x_1, y_2, y_1)$. It is clearly not implied by any of the other conclusion atoms, so Rule 1 – simplifying the conclusion to its core – will not help. But maybe it is produced by another dependency:

Rule 5: Remove atoms from the conclusion, if it they are implied
applied to: $L(x_1, x_2, x_2) \rightarrow P(x_1, y_2, y_1) \wedge Q(y_1, y_3, x_2) \wedge Q(\mathsf{a}, y_3, x_2)$ (σ''_3)

More formally, by *implied*, the following is meant: If dependency τ' is produced from τ by removing atoms from the conclusion, then the removed atoms are implied, if $(\Sigma \setminus \{\tau\}) \cup \{\tau'\}$ is logically equivalent to Σ. Indeed, looking at σ'_1, we see that it produces the atom $P(x_1, y_1, \mathsf{a})$ under more general antecedents. This clearly implies our first atom $P(x_1, y_2, y_1)$, so we can drop that. Furthermore, we see that the second conclusion atom $Q(y_1, y_3, x_2)$ is implied by the last one $Q(\mathsf{a}, y_3, x_2)$ and we can also drop it.

Let us look at the one dependency we did not rewrite up to now, σ_2: The first conclusion atom $P(x_1, y_1, y_2)$ is a more specific case of $P(x_1, y_1, \mathsf{a})$ from σ'_1, and the other atoms are also not very different from those implied by some dependencies. Let us check whether σ_2 is needed at all:

Rule 4: Remove the following dependency, if it is implied by other dependencies
applied to: $L(x_1, x_1, x_1) \rightarrow P(x_1, y_1, y_2) \wedge Q(y_2, y_3, x_1) \wedge R(y_1, x_1, y_2)$ (σ_2)

We see that the last atom $R(y_1, x_1, y_2)$ is implied by $R(y_1, x_2, \mathsf{a})$ of σ_1, and the remaining one $Q(y_2, y_3, x_1)$ by σ'''_3. So indeed, we can remove σ_2. Our mapping is now given as follows:

- $L(x_1, x_2, x_3) \rightarrow P(x_1, y_1, \mathsf{a}) \wedge R(y_1, x_2, \mathsf{a})$ (σ'_1)
- $L(x_1, x_2, x_2) \rightarrow Q(\mathsf{a}, y_3, x_2)$ (σ'''_3)
- $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \rightarrow R(x_2, y_4, x_3)$ (σ'_4)

It can be checked that this mapping is now minimal according to our optimality criteria. ◀

1. **Simplify the conclusion** to its core
2. **Simplify the antecedent** to its core
3. **Split** the dependency if possible
4. **Remove** the dependency, if it is implied by other dependencies
5. **Remove atoms** from the conclusion, if it they are implied by other dependencies

Figure 4 Rewrite rules for optimization and normalization (informal formulation).

Indeed, these rules are sufficient for achieving all four optimality criteria for mappings based on s-t tgds among split-reduced mappings. Even more interesting, they actually yield a unique normal form. Let us now summarize the rewriting system as defined by [14]:
Note that the first and second rules come down to core computations. If the length of each dependency is bounded by a constant, this can be done efficiently (cf. [13]). The last two items are implication tests, which are well known to be efficiently computable [2]. Splitting can also be efficiently performed. In total, the normal form can be computed in polynomial time if the length of each dependency is bounded by a constant [14].

5.2 Summary

In this section, we have seen how to compute an optimized unique normal form of a schema mapping based on s-t tgds. The resulting schema mapping is cardinality-, antecedent-, subset- and variable-minimal among all split-reduced schema mappings.

Through a quite complex extension, and a reformulation of what "split-reduced" should mean in the presence of egds, an optimized but not normalized normal form can be obtained for schema mappings based on s-t tgds and target egds [14].

6 Decidability of reasoning with relaxed notions of equivalence

In the previous section we have seen how to obtain optimized schema mappings under logical equivalence. However, as we have talked about in the beginning, logical equivalence is quite restrictive in the optimization potential it admits. In this section, we will therefore discuss the question of the computational properties of optimality under relaxed notions of equivalence.

We have seen that logical-, DE- and CQ-equivalence form a hierarchy where each notion might offer additional optimization potential compared to the one before. That is, while logical equivalence is the most restrictive, CQ-equivalence is the least restrictive.

A number of questions remained. The first one is:

- For which classes of schema mappings is this hierarchy proper, that is, for which schema mappings can we really gain **additional optimization potential**? The counterpart of this question is one about computability and complexity:
- If there is additional optimization potential, can we find algorithms for reasoning about them? In particular, can we construct **algorithms for finding optimal** schema mappings given various optimality criteria?

These are the questions that will guide this section. They will lead us to the boundaries of computability as explored by Fagin et al. [9] and Pichler et al. [19].

Before looking at questions of computability, let us first find out for which classes of schema mappings there is additional optimization possible. In other words, when is the hierarchy of logical-, DE- and CQ-equivalence strict, and when does it collapse?

6.1 Hierarchy or collapse

Let us first summarize what we have seen earlier in Section 3 when we introduced logical equivalence, DE-equivalence and CQ-equivalence:

- Logical equivalence, DE-equivalence and CQ-equivalence form a hierarchy (Proposition 8): $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}' \Rightarrow \mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}' \Rightarrow \mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$.
- For schema mappings based on s-t tgds and target tgds, all three notions are distinct (through Examples 3 and 5 in Section 3).

Similar examples show that the three notions are different also for s-t tgds with target tgds [9] or target egds [14] as well as for SO tgds [9]. By contrast, we will now see that for schema mappings based on s-t tgds, the three notions actually coincide. We thus return to an example we have seen in a similar form before (Example 5). This time, it will allow us to find out something quite different:

▶ **Example 26.** For the source schema $S = \{P\}$ and the target schema $T = \{Q\}$ let the schema mapping $\mathcal{M}$ be defined by the following dependencies:

- $P(x, y) \to Q(x, x)$
- $Q(x, y) \to x = y$

Let us compare this to the schema mapping based on just the s-t tgd. That is, we define the schema mapping $\mathcal{M}'$ based on

- $P(x, y) \to Q(x, x)$

We have that the schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are CQ-equivalent, but not DE-equivalent. As an intuition for the CQ-equivalence, observe that the egd has no effect on the cores of the universal solutions, since the atoms contained in it already have the form $Q(x, x)$.

Now let $I = P(\mathsf{a}, \mathsf{a})$. The canonical universal solution, which in this case is also the core of the universal solutions, will be $J = \{Q(\mathsf{a}, \mathsf{a})\}$ for both $\mathcal{M}$ and $\mathcal{M}'$. But now consider $J' = \{Q(\mathsf{a}, \mathsf{a}), Q(u, v)\}$, for variables u and v. This instance is universal for both $\mathcal{M}$ and $\mathcal{M}'$, evidenced by the homomorphism $[u \mapsto a, v \mapsto a]$. Still, while J' is a universal solution for $\mathcal{M}'$, it is not even a solution for $\mathcal{M}$. ◀

Fagin et al. [9] identify this property as the key reason for CQ-equivalence and DE-equivalence to be distinct for a class of schema mappings: For two DE-equivalent schema mappings, there is a universal solution for one mapping that is not even a solution for the other mapping.

▶ **Definition 27.** [9] Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping. $\mathcal{M}$ has *all the universal solutions*, if whenever

$$J \in \mathsf{UnivSol}(I, \mathcal{M}) \text{ and } J \leftrightarrow J', \text{ then } J' \in \mathsf{Sol}(I, \mathcal{M})$$ ◀

Recall that $\leftrightarrow$ denotes homomorphic equivalence in this context. That is, for this property, every instance homomorphically equivalent to a universal solution must also be a universal solution. Note that from $J' \in \mathsf{Sol}(I, \mathcal{M}')$ through homomorphic equivalence to J follows that $J' \in \mathsf{UnivSol}(I, \mathcal{M}')$. Given this definition, we then know that:

▶ **Proposition 28.** *[9] If $\mathcal{M}$ and $\mathcal{M}'$ have all the universal solutions, then $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$ implies $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$.* ◀

That is, if all mappings in a class of schema mappings have all the universal solutions, then for this class DE- and CQ-equivalence coincide. Also, the following sufficient condition for this property is attained:

▶ **Definition 29.** [9] Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping. $\mathcal{M}$ is *preserved under target homomorphisms*, if whenever

$$J \in \mathsf{Sol}(I, \mathcal{M}) \text{ and } J \to J', \text{ then } J' \in \mathsf{Sol}(I, \mathcal{M})$$ ◀

This property holds for schema mappings based on s-t tgds. Knowing that preservation under target homomorphism holds has the following important consequence:

▶ **Proposition 30.** *[9] If $\mathcal{M}$ and $\mathcal{M}'$ are*
- *both preserved under target homomorphisms, and*
- *both have that $\mathsf{Sol}(I, \mathcal{M}) \neq \emptyset$ implies $\mathsf{UnivSol}(I, \mathcal{M}) \neq \emptyset$ for all I*

then $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$ iff $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$ iff $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$. ◀

From this, since the properties hold for s-t tgds, we have that the three notions of logical-, DE- and CQ-equivalence coincide for mappings based on s-t tgds. Also, since the three notions are distinct for the classes of schema mappings
- based on s-t tgds and target tgds
- based on s-t tgds and target egds
- based on SO tgds

we know through the preceding theorem that they are *not* generally preserved under target homomorphisms for these classes. In total, if we allow target egds, target tgds or SO dependencies, then the relaxed notions of equivalence offer additional optimization potential.

6.2 Decidability of equivalence and optimization

First, let us note that for mappings based on s-t tgds (where logical, DE-, and CQ-equivalence coincide), all three notions are decidable. The decidability proof comes down to checking implication of dependencies by the chase (cf. e.g. [7]).

▶ **Proposition 31.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be schema mappings based on s-t tgds. Then it is decidable whether $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$, $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$ and $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$.* ◀

So in this case, the problems are decidable, but there is no additional optimization power, since the notions coincide. Given that for a variety of schema mapping classes there is clearly additional optimization power using DE- and CQ- equivalence, we would like to use this power by appropriate algorithms for reasoning about them. In [9], the following general bounds are shown:

▶ **Theorem 32.** *[9] Given two schema mappings $\mathcal{M}$ and $\mathcal{M}'$ based on* s-t tgds and a weakly acyclic set of target tgds
- *it is* decidable *whether $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$*
- *it is* undecidable *whether $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$*

The undecidability results holds even for copy s-t tgds and full target tgds. ◀

The decidability proof again comes down to checking implication of dependencies by the chase. The undecidability result is based on a reduction from Datalog equivalence (which is undecidable as shown in [20]), where target tgds are used to mirror recursive Datalog rules.

Further exploration of these bounds is done by Pichler et al. in [19], where schema mappings based on target egds and target tgds are considered under DE-equivalence in addition to CQ-equivalence. Also, various optimality criteria are discussed there.

▶ **Theorem 33.** *[19] For schema mappings $\mathcal{M}$ and $\mathcal{M}'$ based on* full s-t tgds and full target tgds*, or* s-t tgds and target egds*, the following problems are* undecidable

- *$\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$ and $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$*
- *σ-redundancy of $\mathcal{M}$ w.r.t. DE- and CQ-equivalence*
- *subset-minimality of $\mathcal{M}$ w.r.t. DE- and CQ-equivalence*
- *cardinality-minimality of $\mathcal{M}$ w.r.t. DE- and CQ-equivalence* ◀

Thus leaving out target dependencies leads to a collapse of the hierarchy, hence no additional optimization power. Adding target dependencies leads to undecidability. These bounds leave one possibility open: If we require the target dependencies to be fixed, can we then optimize the s-t tgds further?

Towards this goal, the following connection between normalization of Section 5 and relaxed notions of equivalence was shown for s-t tgds and target tgds or egds. Here we consider source egds, which are simply egds defined over the source schema.

▶ **Theorem 34.** *[19] Let $\mathcal{M}$ and $\mathcal{M}'$ be schema mappings based on* s-t tgds and target tgds or egds*. Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ the sets of s-t tgds resp. target dependencies of $\mathcal{M}$. Assume the same for Σ' and $\mathcal{M}'$.*
If $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$, then there exists a common set Σ_s^* *and* Σ_{st}^* *of* source egds resp. s-t tgds*, s.t.*

$$\Sigma \equiv_{\mathsf{log}} \Sigma_s^* \cup \Sigma_{st}^* \cup \Sigma_t \text{ and } \Sigma' \equiv_{\mathsf{log}} \Sigma_s^* \cup \Sigma_{st}^* \cup \Sigma_t' \quad \blacktriangleleft$$

So why is this theorem interesting? Assume that we have CQ-equivalent schema mappings whose target dependencies are logically equivalent. Given the previous theorem, we can normalize the source and s-t tgds upholding logical equivalence. But given that also the target dependencies are logically equivalent, the two schema mappings are altogether logically equivalent. This has the following consequence:

▶ **Theorem 35.** *[19] In the same setting as in Theorem 34. If $\Sigma_t \equiv_{\mathsf{log}} \Sigma_t'$, then*

$$\Sigma \equiv_{\mathsf{log}} \Sigma' \text{ iff } \Sigma \equiv_{\mathsf{DE}} \Sigma' \text{ iff } \Sigma \equiv_{\mathsf{CQ}} \Sigma' \quad \blacktriangleleft$$

This settles the question of whether optimization is possible if the target dependencies are fixed: There is no additional optimization power.
The third road to finding interesting decidable fragments is to look at special cases, e.g. mappings based on functional dependencies or inclusion dependencies. However, the following result weakens this hope for CQ-equivalence:

▶ **Theorem 36.** *[19] CQ-equivalence is* undecidable *for schema mappings based on s-t tgds and* at most one key dependency per target relation. ◀

Interestingly, the situation for DE-equivalence looks quite different:

▶ **Theorem 37.** *[19] DE-equivalence is* decidable *for schema mappings based on s-t tgds and weakly acyclic sets of* functional- and inclusion dependencies *as target dependencies.* ◀

These two results show our first disparity between the computational properties of DE- and CQ-equivalence.

We have now talked about various optimization tasks under both DE- and CQ-equivalence. From the point of schema mappings, we looked at those based on s-t tgds only, and at those with target egds or target tgds in addition to s-t tgds. What we still have not discussed are SO tgds. Here the following results are known:

▶ **Theorem 38.** *[12] Let $\mathcal{M}$ and $\mathcal{M}'$ be given by SO tgds. It is undecidable whether*
- *$\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$*
- *$\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$ even if it is known that $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$*
- *$\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$ for mappings based on SO tgds and source key deps.* ◀

The proof of the first bullet is based on results about the existence of inverses from [1], about which we will talk a bit later in this chapter.

The previous theorem talks about equivalence between schema mappings based on SO tgds (showing undecidability) and before we talked about equivalence between schema mappings based on s-t tgds (yielding decidability). Recently, Fagin and Kolaitis [7] looked at equivalence where one mapping is based on SO tgds and the other one is based on s-t tgds:

▶ **Theorem 39.** *[7] Let $\mathcal{M}$ be given by SO tgds and $\mathcal{M}'$ be given by s-t tgds*
- *it is* undecidable *whether $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$*
- *it is* decidable *whether $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$* ◀

Interestingly, CQ-equivalence becomes decidable in this case, while it is undecidable for schema mappings based on SO tgds plus source key dependencies.

6.3 Equivalence to classes of schema mappings

In this final subsection, we will talk about the computational properties of deciding whether a mapping from some class of schema mappings is equivalent to a mapping in a more restricted class of schema mappings.

Results in this area were shown in [9] for the following problem: Given a schema mapping, under which conditions is it CQ-equivalent to a mapping consisting of s-t tgds?

We start with mappings based on full s-t tgds and full target tgds. For this, we need the following concept: A mapping has *bounded parallel chase* if there is a constant, such that for every source instance, the parallel chase needs at most that constant number of steps. Using this, we have that

▶ **Theorem 40.** *[9] Let $\mathcal{M}$ be a schema mapping based on* full s-t tgds and full target tgds*. There exists a schema mapping $\mathcal{M}'$ based on* full s-t tgds *with $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$ iff $\mathcal{M}$ has bounded parallel chase.* ◀

The problem of finding such a schema mapping is undecidable [9]. We now look at mappings based on SO tgds. Again, we need a characterizing concept, but this time it is more complex:

▶ **Definition 41.** [9] The *Gaifman graph of facts G* of a target instance K is the graph whose nodes are the facts of K and there is an edge between two facts if they have a null in common. A *fact block* (f-block) of K is a connected component of G.

Mapping $\mathcal{M}$ has *bounded f-block size* if there is a constant such that $\mathsf{core}(I, \mathcal{M})$ has f-block size bounded by this constant for every source instance I. ◀

In [7], it was shown that deciding whether the f-block size of an SO tgd is bounded by a given number is equivalent to a problem we have already seen in the previous section: is a schema mapping based on s-t tgds equivalent to one based on SO tgds. Importantly, the notion of bounded f-block size characterizes CQ-equivalence of an SO tgd to s-t tgds:

▶ **Theorem 42.** *[9] Let $\mathcal{M}$ be a schema mapping based on an SO tgd. There exists a mapping $\mathcal{M}'$ based on s-t tgds with $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$ iff $\mathcal{M}$ has bounded f-block size.* ◀

Note that in [9] a characterization is also given for schema mappings based on s-t tgds and target tgds with terminating chase.

Concerning schema mapping languages that can express only a subset of s-t tgds, results where shown by ten Cate and Kolaitis [21]. Let a LAV tgd be a tgd with a single atom on the left-hand side (this corresponds to what is called "extended LAV" in [7], compared to "strict LAV" which requires that all variables on the left-hand side occur just once).

▶ **Theorem 43.** *For a given schema mapping $\mathcal{M}$, deciding whether there exists a schema mapping $\mathcal{M}'$ such that $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$ is* NP-complete *if*

- *$\mathcal{M}$ is given by* s-t tgds *and $\mathcal{M}'$ shall be definable by* full s-t tgds
- *$\mathcal{M}$ is given by* s-t tgds *and $\mathcal{M}'$ shall be definable by* LAV s-t tgds
- *$\mathcal{M}$ is given by* LAV s-t tgds *and $\mathcal{M}'$ shall be definable by* full s-t tgds

it is decidable in polynomial time *if*

- *$\mathcal{M}$ is given by* full s-t tgds *and $\mathcal{M}'$ shall be definable by* LAV s-t tgds ◀

6.4 Summary

In this section, we have looked into the power of logical-, DE- and CQ-equivalence for optimizing schema mappings. We have seen that for schema mappings based on s-t tgds only, all three notions of equivalence coincide and therefore admit no additional potential for optimization. We then looked at schema mappings based on target egds or target tgds in addition to s-t tgds, as well as SO tgds. We have seen that there clearly is additional potential for optimization using relaxed notions of equivalence.

However, unfortunately, most tasks are undecidable in general apart from logical equivalence for s-t tgds and weakly acyclic sets of target tgds. In particular, DE- and CQ-equivalence are undecidable for schema mappings based on s-t tgds and target tgds or target egds. For SO tgds, even logical equivalence is undecidable. We also looked at how to find mappings which are CQ-equivalent to more restricted classes of schema mappings.

Altogether, many of the boundaries are known, but decidable cases are still sparse. Still, the decidable special case for DE-equivalence and schema mappings with functional and inclusion dependencies shows an interesting disparity between DE- and CQ-equivalence. We summarize the results in Figure 5.

7 Equivalence and optimality for schema mapping management

In the previous two sections, we first discussed logical equivalence and its application to normalization. After that, we talked about DE- and CQ-equivalence and the boundaries of computability. In this section, we will talk about the notions of equivalence we have not covered up to this point: Schema mappings which are equivalent w.r.t. the source information transferred (S-equivalence) or equivalent w.r.t. the target information covered (T-equivalence). They were introduced and applied by Arenas et al. [1].

These notions have a slightly different character compared to logical-, DE- and CQ-equivalence. In particular, we can use them to compare schema mappings that have different source or target schemas. On the other hand, mappings which are e.g. S-equivalent may produce completely different target instances. Still, they guarantee that we can reconstruct the original target instance using some other schema mapping.

Our focus in this section will be on applications of S-equivalence and T-equivalence, in particular combined with the corresponding redundancy notions of source-redundancy and

Optimization potential

Logical-, DE- and CQ-equivalence **coincide** for
- mappings based on s-t tgds

They are **distinct** for
- mappings based on s-t tgds and target tgds or target egds
- mappings based on SO tgds

Boundaries of decidability

Logical equivalence is
- decidable for mappings based on s-t tgds and sets of weakly acyclic target tgds
- undecidable for mappings based on SO tgds

Data-exchange equivalence is
- undecidable for mappings based on s-t tgds, target tgds or target egds
- decidable for mappings with weakly acyclic sets of functional- and inclusion deps.

Conjunctive-query equivalence is
- undecidable for mappings based on s-t tgds, target tgds or target egds (even if restricted to a single key dependency per relation)
- undecidable for mappings based on SO tgds and source key dependencies
- decidable if one mapping is given by SO tgds, the other one by s-t tgds

Further results

Optimality is undecidable for mappings based on s-t tgds, target tgds or egds for
- σ-redundancy, subset-minimality, cardinality-minimality

CQ-equivalence to mappings based on s-t tgds is characterized for
- mappings based on target tgds with terminating chase
- mappings based on SO tgds

Figure 5 Optimization potential and boundaries of decidability.

target-redundancy. We will see that for the important area of characterizing the operators for schema mapping management, these notions of equivalence and optimality find natural applications. After that, we will briefly discuss some algorithmic properties.

7.1 Application to schema mapping management

We begin by talking about the *extract* operator of schema mapping management [18]. Though there are a number of possible characterizations, the intended meaning of the extract operator is the following: Given a schema mapping, find a new source schema that captures exactly the information that participates in it. Let us start by looking at an example.

▶ **Example 44** (based on [1]). Over the source schema $S = \{P, Q, R\}$ and the target schema $T = \{U, V, W\}$ let the schema mapping $\mathcal{M}$ be given by the following dependencies:
- $P(x, y) \to \exists u\, W(x, u) \wedge U(x, x)$ (σ_1)
- $P(x, y) \wedge R(y, z) \to \exists v\, V(x, y, v)$ (σ_2)

Before taking an in-depth look into what the dependencies of schema mapping $\mathcal{M}$ do, let us look at the source relation Q. It actually never occurs in the dependencies of $\mathcal{M}$.

We could *extract* a new source schema S' that does not include the information of Q at all and then use two new schema mappings: $\mathcal{M}_1$ from S to S' migrates from the old source schema to the new one. $\mathcal{M}_2$ from S' to T uses this new source schema to map to the target schema. In this example, we could take $\mathcal{M}_1$ to be just copy tgds from $S = \{P, Q, R\}$ to $S' = \{P, Q\}$ i.e. based on

- $P(x, y) \to P'(x, y)$
- $R(x, y) \to R'(x, y)$

and $\mathcal{M}_2$ to consist exactly of our two original dependencies σ_1 and σ_2 adapted to the new schema. Here, we would trivially have that the two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ composed do the same thing as the original $\mathcal{M}$, or more precisely $\mathcal{M}_1 \circ \mathcal{M}_2 \equiv_{\mathsf{log}} \mathcal{M}$. ◀

In the preceding example, we have extracted a new source schema S' and $\mathcal{M}_1$ and $\mathcal{M}_2$ so that the composition $\mathcal{M}_1 \circ \mathcal{M}_2$ yields the original mapping $\mathcal{M}$. This feels like a natural condition for an extract operator, but as we have seen, this condition alone yields rather unimpressive results for S', $\mathcal{M}_1$ and $\mathcal{M}_2$. Let us continue our example.

▶ **Example 44 (ctd).** In our previous example, we saw that we may find unsatisfying $\mathcal{M}_1$ and $\mathcal{M}_2$ s.t. $\mathcal{M}_1 \circ \mathcal{M}_2 \equiv_{\mathsf{log}} \mathcal{M}$, if we impose no further restrictions. Yet indeed, we want to extract *exactly* the information that participates in $\mathcal{M}$. Let us look at the source information that participates in $\mathcal{M}$, and should therefore transferred by $\mathcal{M}_1$.

In our first dependency σ_1, only the first variable x is actually used in the conclusion. We must not transfer y if we want to capture exactly the needed information. Similarly for σ_2, only the result after the join between P and R is relevant. In particular, we do not need the variable z in the conclusion. Let us express this as a schema mapping $\mathcal{M}_1$ based on

- $P(x, y) \to P_1(x)$
- $P(x, y) \wedge R(y, z) \to P_2(x, y)$

To make our argument about "transferring the needed amount of source information" precise, we have a tool at our hand: S-equivalence expresses that two schema mappings transfer the same amount of source information. That is, we have found an $\mathcal{M}_1$ with $\mathcal{M}_1 \equiv_{\mathsf{S}} \mathcal{M}$.

Now having constructed $\mathcal{M}_1$, let us talk about $\mathcal{M}_2$, which should be able to do two things. First, it should be able to yield $\mathcal{M}$ in the sense that $\mathcal{M}_1 \circ \mathcal{M}_2 \equiv_{\mathsf{log}} \mathcal{M}$. Secondly, we should require that $\mathcal{M}_2$ really covers exactly the amount of target information needed, or in other words $\mathcal{M}_2 \equiv_{\mathsf{T}} \mathcal{M}$. Let us construct such an $\mathcal{M}_2$ based on

- $P_1(x) \to \exists u\, W(x, u) \wedge U(x, x)$
- $P_2(x, y) \to \exists v\, V(x, y, v)$

Altogether, we now have found schema mappings that capture exactly the information participating in $\mathcal{M}$, by requiring $\mathcal{M}_1 \equiv_{\mathsf{S}} \mathcal{M}$ and $\mathcal{M}_2 \equiv_{\mathsf{T}} \mathcal{M}$. ◀

Our characterization of how we expect the *extract* operator to behave is now reasonably complete. However while $\mathcal{M}_1$ transfers exactly the information needed, and $\mathcal{M}_2$ covers the information needed, the in-between schema S' has no condition imposed on it so far. In particular, we could still store every x from $P(x, y)$ as $P_1(x, x, x, x)$, which intuitively is not a good result of the extract operator. Let us continue our example.

▶ **Example 44 (ctd).** The problem we still have is possible redundancy in the new source schema S'. While using $P_1(x, x, x, x)$ as an intermediate atom seems quite drastic, actually we also have a somewhat surprising redundancy in $\mathcal{M}_1$, S' and $\mathcal{M}_2$ from the previous example, though it is a bit subtle:

Let us look at the source instance $I = \{P(\mathsf{a}, \mathsf{a}), P(\mathsf{b}, \mathsf{b}), R(\mathsf{b}, \mathsf{b})\}$. The canonical universal solution of I using our schema mapping $\mathcal{M}_1$ is $J = \{P_1(\mathsf{a}), P_1(\mathsf{b}), P_2(\mathsf{b}, \mathsf{b})\}$. But actually we

do not need to store $P(\mathsf{b})$, since it occurs as $P_2(\mathsf{b},\mathsf{b})$ anyway. That is, any x from $P(x,y)$ that has a join-partner in R need not necessarily be stored in the intermediate instance. So the kind of redundancy we are talking about here is redundancy of possible *source instances or target instances*.

Of course, our current schema mapping $\mathcal{M}_2$ would not be sufficient to make use of this information, it would require a mapping with an additional dependency of the form $P_2(x,y) \to \exists u\, W(x,u) \wedge U(x,x)$. Yet this modified schema mapping is clearly T-equivalent, it covers the same target information. Let us avoid this redundancy in another way. Add to *both* $\mathcal{M}_1$ and $\mathcal{M}_2$ the following dependency:

- $P_2(x,y) \to P_1(x)$

yielding $\mathcal{M}_1'$ and $\mathcal{M}_2'$. This is a target tgd for $\mathcal{M}_1'$ and a source tgd for $\mathcal{M}_2'$.

Let us sum up and make precise our argument about redundancy. We are talking about possibly redundant instances here, and the optimality notions appropriate for this case have been already introduced in Section 4: source-redundancy and target-redundancy. We would like $\mathcal{M}_1$ to be target non-redundant among the S-equivalent mappings, and $\mathcal{M}_2$ to be source non-redundant among the T-equivalent mappings. ◀

Through a progression of examples, we have now identified a characterization for the *extract* operator that meets natural requirements. Let us make this definition explicit:

▶ **Definition 45.** [1] Let $\mathcal{M} = (S,T,\Sigma)$ be a mapping. $(\mathcal{M}_1,\mathcal{M}_2)$ is an extract of $\mathcal{M}$ if

- $\mathcal{M}_1 \circ \mathcal{M}_2 \equiv_{\mathsf{log}} \mathcal{M}$
- $\mathcal{M}_1 \equiv_{\mathsf{S}} \mathcal{M}$ and $\mathcal{M}_1$ is target non-redundant w.r.t. $\equiv_{\mathsf{S}}$
- $\mathcal{M}_2 \equiv_{\mathsf{T}} \mathcal{M}$ and $\mathcal{M}_2$ is source non-redundant w.r.t. $\equiv_{\mathsf{T}}$ ◀

We finish this subsection on the extract-operator by a few notes: Apart from the characterization, there also exists an algorithm for computing extracts for mappings based on s-t tgds with FO formulas in the antecedent. It is based on rewriting and composition [1]. The exact language needed to express these extracts is still open.

Apart from the extract operator, the merge operator was analyzed in [1] as well as the setting of schema evolution. Also, a characterization of the inverse operator [6] is given. The inverse operator is discussed in detail in Chapter 3 of this book.

7.2 Decidability and complexity

For the following results about the properties of our notions, we will be mainly talking about the ordering relations $\preceq_{\mathsf{S}}$ and $\preceq_{\mathsf{T}}$ instead of the equivalence relations $\equiv_{\mathsf{S}}$ and $\equiv_{\mathsf{T}}$. Our first goal will be to find algorithms for deciding these ordering relation.

An alternative characterization based on the following notions brings us one step closer to this goal: A query Q over source schema S is called *target rewritable* under $\mathcal{M}$, if there is a query Q' over T such that $Q(I) = \mathsf{certain}(Q,I,\mathcal{M})$ for all I. Then we have:

▶ **Theorem 46.** *[1] Let $\mathcal{M} = (S,T,\Sigma)$ and $\mathcal{M}' = (S,T',\Sigma')$ be a schema mapping based on s-t tgds. Then $\mathcal{M} \preceq_{\mathsf{S}} \mathcal{M}'$ iff for every query Q, if Q is target rewritable in $\mathcal{M}$ then Q is target rewritable in $\mathcal{M}'$. This result even holds if FO formulas are allowed as antecedents.* ◀

Backed up by this result we see that, equivalence and ordering w.r.t. source information transferred (which are based on transferring enough source information to be able to recover the original target information) are both intuitively and provably close to target rewritability.

For schema mappings based on s-t tgds, even with inequivalence in the antecedent, deciding $\mathcal{M} \preceq_S \mathcal{M}'$ is in coNEXPTIME. However, for schema mappings based on s-t tgds that allow FO formulas as antecedents, it is undecidable whether $\mathcal{M} \preceq_S \mathcal{M}'$ [1].

7.3 Summary

In this section, we looked at S-equivalence and T-equivalence. We applied them to characterize the operator *extract*, one of the central operators of schema mapping management. To characterize this operator, we saw that notions of equivalence (S- and T-equivalence) were needed, as well as notions of optimality (source- and target-redundancy).

We hinted at other operators *invert*, *merge*, and the setting of schema evolution that can be characterized using these concepts. For all of these, characterizations and algorithms can be found in [1]. We finished this section by a quick look at questions of decidability and complexity.

8 Reasoning in the broader sense

In the previous sections, we discussed reasoning about schema mappings in a strict sense. Our topics were primarily equivalence and optimality. But of course the term "reasoning" can be applied to a broad range of important tasks associated with schema mappings. While we cannot cover all of them, we want to finish this chapter by at least talking about one of them, in particular one that fits very well into what we discussed up to now.

Reasoning about schema mappings as a task humans have to do poses a number of challenges. Of course, using optimization techniques beforehand might help create schema mappings that are easier to handle as humans. Yet at some point, we have to deal with the actual schema mappings we have at that moment.

Given such a schema mapping, one of the first challenges is to find out what this schema mapping actually *does*, or rather what its intended meaning is. The other question that usually enters our reasoning process earlier than we might like is what a certain schema mapping *does wrong*. This topic of finding errors, that is debugging schema mappings, will be our topic in this section.

8.1 Analyzing and debugging with routes

Since we want to actually understand and debug some arbitrary schema mapping that we are given, let us start with such an example.

▶ **Example 47** (based on [5])**.** In this example, to ease debugging, we leave the names of the relation symbols intact. Also, universally quantified variables are denoted by words starting in lower case (sal), existentially quantified variables are given starting with upper case (M) and constants as usual in sans-serif font (Smith or 6689).

Let the mapping $\mathcal{M}$ over the source schema $\mathsf{ManhattenCredit} = \{\mathsf{Cards}, \mathsf{SuppCards}\}$ and the target schema $\mathsf{FargoFinance} = \{\mathsf{Accounts}, \mathsf{Clients}\}$ be given by the following dependencies:

- $\mathsf{Cards}(cn, l, s, n, m, sal, loc) \to \exists A\,(\mathsf{Accounts}(cn, l, s) \wedge \mathsf{Clients}(s, m, m, sal, A))$ (σ_1)
- $\mathsf{SuppCards}(an, s, n, a) \to \exists M, I\ \mathsf{Clients}(s, n, M, I, a)$ (σ_2)
- $\mathsf{Accounts}(a, l, s) \to \exists N, M, I, A\ \mathsf{Clients}(s, N, M, I, A)$ (σ_3)
- $\mathsf{Clients}(s, n, m, i, a) \to \exists N, L\ \mathsf{Accounts}(N, L, s)$ (σ_4)
- $\mathsf{Accounts}(a, I, s) \wedge \mathsf{Accounts}(a', I', s) \to I = I'$ (σ_5)

The schemas and the s-t tgds σ_1 and σ_2 are illustrated in Figure 6. Seemingly, it is a schema mapping describing how data about credit cards is transferred to some financial organization. Without worrying too much for now how this schema mapping actually works in detail, let us try debugging it with a test instance. Let I be given by the following ground atoms:

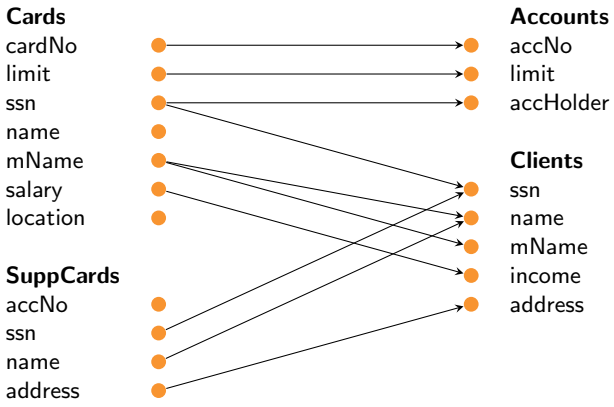

Figure 6 Schemas and s-t tgds of mapping $\mathcal{M}$.

- $\mathsf{Cards}(6689, 15\mathsf{K}, 434, \mathsf{J.Long}, \mathsf{Smith}, 50\mathsf{K}, \mathsf{Seattle})$ (s_1)
- $\mathsf{SuppCards}(6689, 234, \mathsf{A.Long}, \mathsf{California})$ (s_2)

In the target database, we use the following instance J given by:

- $\mathsf{Accounts}(6689, 15\mathsf{K}, 434)$ (t_1)
- $\mathsf{Accounts}(N_1, 50\mathsf{K}, 234)$ (t_2)
- $\mathsf{Clients}(434, \mathsf{Smith}, \mathsf{Smith}, 50\mathsf{K}, A_1)$ (t_3)
- $\mathsf{Clients}(234, \mathsf{A.Long}, M_1, I_1, \mathsf{California})$ (t_4)

Let us take a closer look on t_3, which is slightly strange: In $\mathsf{Clients}(434, \mathsf{Smith}, \mathsf{Smith}, 50\mathsf{K}, A_1)$, why is there a labeled null A_1 introduced, and why does the constant Smith occur twice?

Let us try to answer this question by tracing tuple t_3 back to the atoms that are directly responsible for creating it. We get the following atom s_1 being responsible for creating both t_1 and t_3 through dependency σ_1 using homomorphism h:

- $\mathsf{Cards}(6689, 15\mathsf{K}, 434, \mathsf{J.Long}, \mathsf{Smith}, 50\mathsf{K}, \mathsf{Seattle})$ (s_1)
  σ_1: $\mathsf{Cards}(cn, l, s, n, m, sal, loc) \rightarrow \exists A\,(\mathsf{Accounts}(cn, l, s) \wedge \mathsf{Clients}(s, m, m, sal, A))$
  h: $\{cn \mapsto 6689, l \mapsto 15\mathsf{K}, s \mapsto 434, n \mapsto \mathsf{J.Long}, m \mapsto \mathsf{Smith}, sal \mapsto 50\mathsf{K}, loc \mapsto \mathsf{Seattle}, A \mapsto A_1\}$
- $\mathsf{Accounts}(6689, 15\mathsf{K}, 434)$ (t_1) $\quad$ $\mathsf{Clients}(434, \mathsf{Smith}, \mathsf{Smith}, 50\mathsf{K}, A_1)$ (t_3) ◀

Before using this chase-like step for debugging our schema mapping, let us formally define it:

▶ **Definition 48.** [5] A **satisfaction step** is given as $K_1 \xrightarrow{\sigma,h} K_2$ where

- K_1 is an instance such that $K_1 \subseteq K$ and K satisfies σ
- σ is a tgd $\varphi(\vec{x}) \rightarrow \exists \vec{y}\, \psi(\vec{x}, \vec{y})$
- h is a homomorphism from $\varphi(\vec{x}) \wedge \psi(\vec{x}, \vec{y})$ to K such that h is also a homomorphism from $\varphi(\vec{x})$ to K_1
- K_2 is the *result* of satisfying σ on K_1 with h, where $K_2 = K_1 \cup h(\psi(\vec{x}, \vec{y}))$ ◀

Note that this differs from the definition of a chase step, in particular because the applicability condition is far broader. This is not surprising since such a satisfaction step shall be able to help debug arbitrary solutions, whether they were created through the chase or not. We now continue our debugging:

▶ **Example 47 (ctd).** Let us look at what we can find out using this satisfaction step with the result including t_3. As we can see, the location Seattle is indeed contained in s_1, it just is not copied by σ_1, instead being replaced by a labeled null. This is most probably not the

intended meaning, we will correct it. For the constant Smith occurring twice, we see that σ_1 uses m, m twice in the conclusion, instead of n, m from the antecedent. We can correct this typo as well, modifying σ_1 to

- $\mathsf{Cards}(cn, l, s, n, m, sal, loc) \rightarrow \mathsf{Accounts}(cn, l, s) \wedge \mathsf{Clients}(s, n, m, sal, loc)$ (σ_1')

So we corrected an error, but we might have spotted this error without any help of a debugging system.

Let us look at a more complex case, debugging t_2: $\mathsf{Accounts}(N_1, \mathsf{50K}, 234)$. This atom cannot be traced via a single satisfaction step to source atoms. But it can be traced back to two satisfaction steps (here, we omit the homomorphisms):

- $\mathsf{SuppCards}(6689, 234, \mathsf{A.Long}, \mathsf{California})$ (s_2)
  σ_2: $\mathsf{SuppCards}(an, s, n, a) \rightarrow \exists M, I\ \mathsf{Clients}(s, n, M, I, a)$
- $\mathsf{Clients}(234, \mathsf{A.Long}, M_1, I_1, \mathsf{California})$ (t_4)
  σ_4: $\mathsf{Clients}(s, n, m, i, a) \rightarrow \exists N, L\ \mathsf{Accounts}(N, L, s)$
- $\mathsf{Accounts}(N_1, \mathsf{50K}, 234)$ (t_2) ◀

Before using this sequence of satisfaction-steps for debugging, we again formally describe the notion first:

▶ **Definition 49.** [5] A **route** for J_s with $\mathcal{M}$, I and J is a sequence of satisfaction steps $(I, \emptyset) \xrightarrow{\sigma_1, h_1} (I, J_1) \ldots \xrightarrow{\sigma_n, h_n} (I, J_n)$ where

- J is a solution of I under $\mathcal{M}$
- $J_i \subseteq J$ and σ_i are from $\mathcal{M}$
- $J_s \subseteq J_n$ ◀

Having now defined what we mean by a route, let us use the one we have found in our continuing example for debugging our schema mapping:

▶ **Example 44 (ctd).** The route shows some strange things: The value 50K suddenly appears in t_2, without being required by the dependency. This also shows that J is actually not a *universal* solution, thus we witness the difference between satisfaction step and chase step.

Also, the account contains a labeled null N_1 as the account number, even though in the source tuple, we have the concrete value 6689. The reason is clear looking at this trace: The intermediate atom t_4 simply cannot store this account number. We can correct this by a more complex modification of σ_2 based on a join:

- $\mathsf{Cards}(cn, l, s_1, n_1, m, sal, loc) \wedge \mathsf{SuppCards}(cn, s_2, n_2, a) \rightarrow$
  $\exists M, I\ \mathsf{Clients}(s_2, n_2, M, I, a) \wedge \mathsf{Accounts}(cn, l, s_2)$ (σ_2')

In total, we might not have found all errors through this debugging, but a few obvious ones, and also some non-obvious errors have now been corrected. ◀

To conclude this section, we note a quite important fact for actual debugging with such routes: There is an algorithm for computing a minimal route, essentially in polynomial time w.r.t. the size of the atoms to-be-debugged. Also, there are situations where one route is not enough, but computing all routes is required. There is an algorithm for that as well in [5].

8.2 Summary

In this subsection, we scratched the surface of reasoning about schema mappings in the broader sense. We looked at a particular application of debugging schema mappings using the concept of routes. While we briefly noted algorithmic properties, we had no chance to explore further connections to e.g. the topic of provenance.

This was of course only an exemplified excursion into the broad topic of what "reasoning" about schema mappings may mean. Each of those meanings might fill a chapter of its own.

9 Conclusion

The topic of *reasoning about schema mappings* is a broad one. In this chapter, we focused on some of the central concepts of reasoning tasks: *equivalence* (Section 3) and *optimality* (Section 4). But we also talked about applications and reasoning in the broader sense.

So before summarizing what we discussed in this chapter, let us look at how those topics fit together in how we *actually* reason about schema mappings:

Given a **schema mapping**:
- What does it **do**, and
- Can we find **errors** in it? (*Debugging*, Section 8)
- Can we **optimize** it?
  - automatically using **logical** equivalence? (*Optimizing and Normalizing*, Section 5)
  - is there hope using **relaxed notions**? (*Boundaries of Decidability*, Section 6)
  - or at least preserving the **information** involved? (*Information Transfer*, Section 7)

This process could of course be augmented with any number of other reasoning tasks about schema mappings, both in the strict or in a broader sense. Let us now summarize what we discussed in this chapter:

9.1 Summary

In the first part of this chapter, focused on **concepts**, we introduced notions of *equivalence* and notions of *optimality*. We saw how they naturally arise when working with schema mappings and discussed their relationship to each other. As a quick reference of all involved notions, see Figure 2 and 3 at the ends of Section 3 and 4.

In the second part, we looked at **applications** and **computational properties** of these concepts. We first saw that under *logical equivalence*, one can optimize schema mappings based on s-t tgds under a broad range of optimality criteria, achieving a unique normal form.

We then explored the boundaries of decidability under *data-exchange equivalence* and *conjunctive-query equivalence*. While many of the general problems there are undecidable, we saw that there is both additional potential opened by these relaxations of logical equivalence, as well as some decidable cases that may exploit this additional potential.

We then continued to apply *equivalence in terms of information transfer* to characterizing important operators of schema mapping management. We saw that one can achieve quite concise characterizations in that way.

In the final part, we also gave a glimpse at the **broader sense** of reasoning about schema mappings. There, we briefly looked at analyzing and *debugging schema mappings* by introducing the concept of routes.

9.2 Outlook

While many of the general computational boundaries of reasoning about *equivalence* and *optimality* of schema mappings have been explored, there are a number of theoretical and practical problems open. For practical utilization, the search for useful decidable fragments is paramount. In particular, current algorithms like those illustrated in Section 5 might be extendable to cover an even broader range of schema mappings. Still, new approaches might be needed to cope with relaxed notions of equivalence.

In the broader sense of *reasoning about schema mappings*, we touched only the surface of available material. It is a topic that could fill many chapters of this size.

Acknowledgements. The author wants to thank the editors and reviewers for their exceptionally detailed and helpful comments.

References

1 Marcelo Arenas, Jorge Pérez, Juan L. Reutter, and Cristian Riveros. Foundations of schema mapping management. In *PODS*, pages 227–238, 2010.
2 Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.
3 Philip A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.
4 Philip A. Bernstein and Sergey Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12, 2007.
5 Laura Chiticariu and Wang Chiew Tan. Debugging schema mappings with routes. In *VLDB*, pages 79–90, 2006.
6 Ronald Fagin. Inverting schema mappings. *ACM Trans. Database Syst.*, 32(4), 2007.
7 Ronald Fagin and Phokion G. Kolaitis. Local transformations and conjunctive-query equivalence. In *PODS*, pages 179–190, 2012.
8 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
9 Ronald Fagin, Phokion G. Kolaitis, Alan Nash, and Lucian Popa. Towards a theory of schema-mapping optimization. In *PODS*, pages 33–42, 2008.
10 Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.
11 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.
12 Ingo Feinerer, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. On the undecidability of the equivalence of second-order tuple generating dependencies. In *AMW*, 2011.
13 Georg Gottlob and Alan Nash. Data exchange: computing cores in polynomial time. In *PODS*, pages 40–49, 2006.
14 Georg Gottlob, Reinhard Pichler, and Vadim Savenkov. Normalization and optimization of schema mappings. *VLDB J.*, 20(2):277–302, 2011.
15 Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Data integration: The teenage years. In *VLDB*, pages 9–16, 2006.
16 Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS*, pages 61–75, 2005.
17 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
18 Sergey Melnik. *Generic Model Management: Concepts and Algorithms*, volume 2967 of *Lecture Notes in Computer Science*. Springer, 2004.
19 Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. Relaxed notions of schema mapping equivalence revisited. In *ICDT*, pages 90–101, 2011.
20 Oded Shmueli. Equivalence of datalog queries is undecidable. *J. Log. Program.*, 15(3):231–241, 1993.
21 Balder ten Cate and Phokion G. Kolaitis. Structural characterizations of schema-mapping languages. In *ICDT*, pages 63–72, 2009.