Relaxed Notions of Schema Mapping Equivalence Revisited*

Reinhard Pichler
TU Vienna
pichler@dbai.tuwien.ac.at

Emanuel Sallinger
TU Vienna
sallinger@dbai.tuwien.ac.at

Vadim Savenkov
TU Vienna
savenkov@dbai.tuwien.ac.at

ABSTRACT

Recently, two relaxed notions of equivalence of schema mappings have been introduced, which provide more potential of optimizing schema mappings than logical equivalence: data exchange (DE) equivalence and conjunctive query (CQ) equivalence. In this work, we systematically investigate these notions of equivalence for mappings consisting of s-t tgds and target egds and/or target tgds. We prove that both CQ- and DE-equivalence are undecidable and so are some important optimization tasks (like detecting if some dependency is redundant). However, we also identify an important difference between the two notions of equivalence: CQ-equivalence remains undecidable even if the schema mappings consist of s-t tgds and target dependencies in the form of key dependencies only. In contrast, DE-equivalence is decidable for schema mappings with s-t tgds and target dependencies in the form of functional and inclusion dependencies with terminating chase property.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Relational databases*; H.2.5 [**Database Management**]: Heterogeneous Databases—*Data translation*

1. INTRODUCTION

Schema mappings play an important role in several areas of database research – above all in data integration [16] and data exchange [9]. An important line of research has been concerned with algebraic operations [5, 21] on schema mappings like computing inverses [13, 3, 2] and composing schema mappings [18, 12, 22, 1]. The question of *schema mapping optimization* has been raised only recently. In [10], Fagin et al. laid the foundation for schema mapping optimization by introducing new concepts of "equivalence" between two schema mappings, namely *data exchange equivalence* (DE-equivalence) and *conjunctive query equivalence* (CQ-equivalence). These are natural *relaxations of logical equivalence*. In total, we are thus dealing with three notions of equivalence, which are defined as follows: Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two schema mappings. We say that $\mathcal{M}_1$ and $\mathcal{M}_2$ are

- *logically equivalent* (denoted $\mathcal{M}_1 \equiv \mathcal{M}_2$) iff $\mathcal{M}_1$ and $\mathcal{M}_2$ are satisfied by precisely the same pairs $\langle I, J\rangle$ of source and target instances.
- *DE-equivalent* (denoted $\mathcal{M}_1 \equiv_{DE} \mathcal{M}_2$) iff, for every source instance I, the universal solutions under the mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ coincide.
- *CQ-equivalent* (denoted $\mathcal{M}_1 \equiv_{CQ} \mathcal{M}_2$) iff, for every source instance I, any conjunctive query posed against the target schema yields the same certain answers for the mappings $\mathcal{M}_1$ and $\mathcal{M}_2$. In [10], also an alternative criterion was proved, namely: $\mathcal{M}_1 \equiv_{CQ} \mathcal{M}_2$ iff, for every source instance I, either both $\mathcal{M}_1$ and $\mathcal{M}_2$ have no solution or they both have the same core of the universal solutions.

Formal definitions of the (universal) solutions and of the core will be recalled in Section 2. In [10], the implications $(\mathcal{M}_1 \equiv \mathcal{M}_2) \Rightarrow (\mathcal{M}_1 \equiv_{DE} \mathcal{M}_2) \Rightarrow (\mathcal{M}_1 \equiv_{CQ} \mathcal{M}_2)$ were proved. In general, the converse of neither implication is true.

In this paper, we restrict our attention to schema mappings consisting of source-to-target tuple-generating dependencies (*s-t tgds*) as well as target dependencies in the form of equality-generating dependencies (*target egds*) and/or tuple-generating dependencies (*target tgds*). The following example illustrates that for such mappings, the three notions of equivalence are indeed different and provide different power for detecting the redundancy of dependencies.

EXAMPLE 1.1. Consider the schema mappings $\mathcal{M} = (S, T, \Sigma)$, $\mathcal{M}_1 = (S, T, \Sigma_1)$ and $\mathcal{M}_2 = (S, T, \Sigma_2)$ with source schema $S = \{P\}$, target schema $T = \{Q, R\}$, and sets of dependencies $\Sigma = \{\tau\}, \Sigma_1 = \{\tau, \tau_1\}, \Sigma_2 = \{\tau, \tau_2\}$, s.t. τ, τ_1 and τ_2 are defined as follows:

$$\tau = P(x_1, x_2) \rightarrow Q(x_1, x_1)$$
$$\tau_1 = R(x_1, x_2) \rightarrow R(x_1, x_1)$$
$$\tau_2 = Q(x_1, x_2) \rightarrow Q(x_1, x_1)$$

For $\mathcal{M}_1$, the equivalence $\mathcal{M}_1 \equiv_{DE} \mathcal{M}$ holds (and hence also $\mathcal{M}_1 \equiv_{CQ} \mathcal{M}$). Intuitively, this is due to the fact that τ_1 has no effect on the universal solutions. On the other hand, we have $\mathcal{M}_1 \not\equiv \mathcal{M}$ since, for instance, (I, J) with $I = \{P(a, b)\}$ and $J = \{Q(a, a), R(a, b)\}$ satisfies the mapping $\mathcal{M}$ but not $\mathcal{M}_1$.

For $\mathcal{M}_2$, we have $\mathcal{M}_2 \equiv_{CQ} \mathcal{M}$ but $\mathcal{M}_2 \not\equiv_{DE} \mathcal{M}$ (and hence $\mathcal{M}_2 \not\equiv \mathcal{M}$). Indeed, for any source instance I, the tgd τ_2 has no effect on the core $J^* = \{Q(a, a) \mid (\exists b)P(a, b) \in I\}$ of the universal solutions of I. However, for $I = \{P(a, b)\}$, the target instance $J = \{Q(a, a), Q(y_1, y_2)\}$ where y_1, y_2 are distinct variables, is a universal solution under mapping $\mathcal{M}$ but not under $\mathcal{M}_2$. □

*This work is funded by the Vienna Science and Technology Fund (WWTF), project ICT08-032. Savenkov is supported by the European program "Erasmus Mundus External Cooperation Window".

ICDT 2011, March 21–23, 2011, Uppsala, Sweden

Fagin et al. proved several important properties of *CQ-equivalence* [10]: On the one hand, the authors presented sufficient criteria under which mappings consisting of Second-Order tgds [12] or mappings consisting of s-t tgds and target tgds can be replaced by CQ-equivalent mappings consisting of s-t tgds only. On the other hand, by a straightforward reduction from the equivalence of datalog programs, the authors proved that CQ-equivalence is undecidable for mappings consisting of s-t tgds and full target tgds. In contrast, *DE-equivalence* has been left largely unexplored to date. In particular, it is unclear if DE-equivalence is also undecidable in those cases where CQ-equivalence is undecidable and if DE-equivalence has any potential application to schema mapping optimization. Moreover, so far, neither CQ-equivalence nor DE-equivalence has been considered for mappings containing target egds.

The goal of this work is a systematic investigation of the above recalled relaxed notions of equivalence applied to schema mappings consisting of s-t tgds and target dependencies in the form of egds and/or tgds. Above all, we want to clarify the decidability/undecidability of the relaxed notions of equivalence themselves and of fundamental optimization tasks under these notions of equivalence. We thus first revisit the undecidability proof [10] of CQ-equivalence of mappings with s-t tgds and full target tgds and show that this proof idea can be easily extended to mappings with s-t tgds and target egds. Moreover, also the undecidability of some basic optimization tasks (like detecting if some dependency is redundant) can be easily covered by this approach. However, there are also limits to this approach. In particular, it is unclear how it can be extended to DE-equivalence or how it can be used to establish the decidability/undecidability of important special cases like mappings whose target dependencies are key dependencies only.

We therefore present a different approach to proving the undecidability of these equivalence problems and optimization problems. To this end, we show how one can mimic the computations of a Turing machine by schema mappings and an appropriately chosen sequence of source instances. This will allow us to reduce the Halting problem of Turing machines to the co-problem of DE-equivalence. Actually, this reduction also works for CQ-equivalence. Moreover, this reduction can be extended so as to prove that also further optimization tasks (like deciding if a set of dependencies can be replaced by an equivalent one of smaller cardinality) under both DE-equivalence and CQ-equivalence are undecidable.

We then turn our attention to important special cases of the mappings considered here. This will allow us to identify a significant difference between CQ-equivalence and DE-equivalence. Indeed, by further extending our proof technique via Turing machines, we can prove the undecidability of *CQ-equivalence* for a very restricted class of schema mappings, namely those consisting of s-t tgds and target *key dependencies*. In contrast, we show that *DE-equivalence* is decidable even for a bigger class of mappings, namely mappings consisting of s-t tgds and target dependencies in the form of functional and inclusion dependencies with terminating chase property. Consequently, DE-equivalence is well suited for optimizing such mappings (by detecting redundant dependencies) while CQ-equivalence is not. To the best of our knowledge, this is the first result that underlines the usefulness of DE-equivalence, since all previous approaches to schema mapping optimization were either based on CQ-equivalence [10] or logical equivalence [15].

An inspection of all undecidability proofs in this paper reveals that the undecidability is mainly due to the target dependencies. That is, DE- and CQ-equivalence of two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ remains undecidable even if $\mathcal{M}_1$ and $\mathcal{M}_2$ coincide on the s-t tgds (and only differ on the target dependencies). Likewise, the optimization tasks mentioned above are undecidable even if we only want to simplify the target dependencies. Now what happens if we only want to simplify the s-t tgds? There are essentially two questions to be answered: First, do the relaxed notions of equivalence provide us with additional power (compared with logical equivalence) for natural optimization problems like subset-minimality or cardinality-minimality of the set of s-t tgds? And what about the decidability of the relaxed notions of equivalence of two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ if $\mathcal{M}_1$ and $\mathcal{M}_2$ coincide on the target dependencies (and only differ on the s-t tgds)? Clearly, for mappings consisting of s-t tgds only, no additional simplifications are possible since (as was shown in [10]) DE- and CQ-equivalence coincide with logical equivalence in this case. Of course, for mappings containing also target egds and/or tgds, the situation might change. However, we show that also in the presence of target dependencies, the relaxation of equivalence does not provide us with additional possibilities of simplifying the s-t tgds. Moreover, we show that the DE- and CQ-equivalence of two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ becomes decidable if $\mathcal{M}_1$ and $\mathcal{M}_2$ coincide on the target dependencies.

Organization of the paper and summary of results. In Section 2, we recall some basic notions. A conclusion and an outlook to future work are given in Section 8. Our main results are detailed in the Sections 3 – 7, namely:

- *CQ-equivalence revisited.* In [10], it was shown that CQ-equivalence is undecidable for mappings consisting of s-t tgds and full target tgds. In Section 3, we extend this proof idea so as to show the undecidability of CQ-equivalence for mappings consisting of s-t tgds and target egds. Moreover, we prove the undecidability for some basic optimization tasks under CQ-equivalence.

- *Undecidability via the Halting problem.* In Section 4, we prove also the undecidability of DE-equivalence. The proof is based on a reduction from the Halting problem. This proof idea will then be used in the subsequent sections to derive further undecidability results – both for DE-equivalence and CQ-equivalence.

- *Undecidability of optimization tasks.* Example 1.1 has illustrated that DE- and CQ-equivalence give us additional power (compared with logical equivalence) for natural optimization problems like subset-minimality or cardinality-minimality. In Section 5, we prove the undecidability of these problems.

- *Optimization of the s-t tgds.* In Section 6, we study mappings where the target dependencies are considered as fixed and only the s-t tgds are allowed to vary. We prove the decidability of DE- and CQ-equivalence if the mappings only differ on the s-t tgds. However, we also show that for a broad class of optimization problems, DE- and CQ-equivalence do not give us additional power (compared with logical equivalence).

- *Special cases.* In Section 7, we identify an important difference between DE- and CQ-equivalence: We show that DE-equivalence is decidable if the schema mappings consist of s-t tgds and target dependencies in the form of functional and inclusion dependencies and possess the terminating chase property. In contrast, CQ-equivalence is undecidable even for schema mappings consisting of s-t tgds and target key dependencies.

Due to space limitations, most proofs are only sketched or even omitted. Detailed proofs will be provided in the full paper.

2. PRELIMINARIES

A *schema* $\mathbf{R} = \{R_1, \ldots, R_n\}$ is a set of relation symbols R_i each of a fixed arity. An *instance* I over a schema $\mathbf{R}$ consists of a relation for each relation symbol in $\mathbf{R}$, s.t. both have the same arity. For a relation symbol R, we write I^R to denote the relation of R in I. We only consider finite instances here.

Tuples of the relations may contain two types of *terms*: *constants* and *variables*. The latter are also called *marked nulls* or *labelled nulls*. Two labelled nulls are equal iff they have the same label. For every instance J, we write $dom(J)$, $var(J)$, and $Const(J)$ to denote the set of terms, variables, and constants, respectively, of J. Clearly, $dom(J) = var(J) \cup Const(J)$ and $var(J) \cap Const(J) = \emptyset$. If we have no particular instance J in mind, we write $Const$ to denote the set of all possible constants. We write $\vec{x}$ for a tuple $(x_1, x_2, \ldots, x_n)$. However, by slight abuse of notation, we also refer to the set $\{x_1, \ldots, x_n\}$ as $\vec{x}$. Hence, we may use expressions like $x_i \in \bar{x}$ or $\vec{x} \subseteq X$, etc.

Let $\mathbf{S} = \{S_1, \ldots, S_n\}$ and $\mathbf{T} = \{T_1, \ldots, T_m\}$ be schemas with no relation symbols in common. We call $\mathbf{S}$ the *source schema* and $\mathbf{T}$ the *target schema*. We write $\langle \mathbf{S}, \mathbf{T} \rangle$ to denote the schema $\{S_1, \ldots, S_n, T_1, \ldots, T_m\}$. Instances over $\mathbf{S}$ (resp. $\mathbf{T}$) are called *source* (resp. *target*) *instances*. If I is a source instance and J a target instance, then $\langle I, J \rangle$ is an instance of the schema $\langle \mathbf{S}, \mathbf{T} \rangle$.

Homomorphisms and substitutions. Let I, I' be instances. A *homomorphism* $h\colon I \to I'$ is a mapping $dom(I) \to dom(I')$, s.t. (1) whenever $R(\bar{x}) \in I$, then $R(h(\bar{x})) \in I'$, and (2) for every constant c, $h(c) = c$. If such h exists, we write $I \to I'$. Moreover, if $I \leftrightarrow I'$ then we say that I and I' are *homomorphically equivalent*. In contrast, if $I \to I'$ but not vice versa, we say that I is *more general* than I', and I' is *more specific* than I.

If $h\colon I \to I'$ is invertible, s.t. h^{-1} is a homomorphism from I' to I, then h is called an *isomorphism*, denoted $I \cong I'$. An *endomorphism* is a homomorphism $I \to I$. An endomorphism is *proper* if it is not surjective (for finite instances, this is equivalent to being not injective), i.e., if it reduces the domain of I.

If I is an instance, and $I' \subseteq I$ is such that $I \to I'$ holds but for no other $I'' \subset I'\colon I \to I''$ (that is, I' cannot be further "shrunk" by a proper endomorphism), then I' is called a *core* of I. The core is unique up to isomorphism. Hence, we may speak about *the* core of I. Cores have the following important property: for arbitrary instances J and J', $J \leftrightarrow J'$ *iff* $core(J) \cong core(J')$.

A *substitution* σ is a mapping which sends variables to other domain elements (i.e., variables or constants). We write $\sigma = \{x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n\}$ if σ maps each x_i to a_i and σ is the identity outside $\{x_1, \ldots, x_n\}$. The application of a substitution is usually denoted in postfix notation, e.g.: $x\sigma$ denotes the image of x under σ. For an expression $\varphi(\bar{x})$ (e.g., a conjunctive query with variables in $\vec{x}$), we write $\varphi(\bar{x}\sigma)$ to denote the result of replacing every occurrence of every variable $x \in \bar{x}$ by $x\sigma$.

Schema Mappings and Data Exchange. A *schema mapping* is given by a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ where $\mathbf{S}$ is the source schema, $\mathbf{T}$ is the target schema, and Σ is a set of dependencies expressing the relationship between $\mathbf{S}$ and $\mathbf{T}$ and possibly also local constraints on $\mathbf{S}$ resp. $\mathbf{T}$. The *data exchange problem* associated with $\mathcal{M}$ is the following: Given a (ground) source instance I, find a target instance J, s.t. $\langle I, J \rangle \models \Sigma$. Such a J is called a *solution for* I or, simply, a *solution* if I is clear from the context. The set of all solutions for I under $\mathcal{M}$ is denoted by $Sol(I, \mathcal{M})$. If $J \in Sol(I, \mathcal{M})$ is such that $J \to J'$ holds for any other solution $J' \in Sol(I, \mathcal{M})$, then J is called a *universal solution*. Since the universal solutions for a source instance I are homomorphically equivalent, the core of the universal solutions for I is unique up to isomorphism. It is the smallest universal solution [11]. We write $UnivSol(I, \mathcal{M})$ to denote the set of universal solutions for I under mapping $\mathcal{M}$ and we write $core(I, \mathcal{M})$ to denote the core of the universal solutions.

In the following, we will often identify a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ with the set of dependencies Σ, without explicitly mentioning the schemas, for the sake of brevity.

Embedded dependencies. *Embedded dependencies* [8] over a relational schema $\mathbf{R}$ are first-order formulae of the form $\forall \bar{x}(\varphi(\bar{x}) \to \exists \bar{y}\ \psi(\bar{x}, \bar{y}))$. If $\vec{y}$ is empty, then the dependency is called *full*. In case of *tuple-generating dependencies* (tgds), both *antecedent* φ and *conclusion* ψ are conjunctive queries (CQs) over the relation symbols from $\mathbf{R}$ s.t. all variables in $\vec{x}$ actually do occur in $\varphi(\bar{x})$. *Equality-generating dependencies* (egds) are of the form $\forall \bar{x}\,(\varphi(\bar{x}) \to x_i = x_j)$ with $x_i, x_j \in \bar{x}$. Throughout this paper, we shall omit the universal quantifiers: By convention, all variables occurring in the antecedent are universally quantified over the entire formula. As a further notational convention, we shall write underscores "_" to denote unspecified, fresh variables. For a conjunctive query χ (in the antecedent or the conclusion of some dependency), we write $At(\chi)$ to denote the set of atoms of this CQ.

In the context of data exchange, we are mainly dealing with *source-to-target dependencies* consisting of tuple-generating dependencies (s-t tgds) over the schema $\langle \mathbf{S}, \mathbf{T} \rangle$ (the antecedent is a CQ over $\mathbf{S}$, the conclusion over $\mathbf{T}$) and *target dependencies* in the form of tgds and egds over the schema $\mathbf{T}$. Moreover, in Section 6, we shall also consider *source dependencies* consisting of egds over the schema $\mathbf{S}$ (referred to as "source egds").

Chase. The data exchange problem can be solved by the *chase* [4], a sequence of steps, each enforcing a single constraint within some limited set of tuples. More precisely, let Σ contain a tgd $\tau\colon \varphi(\bar{x}) \to (\exists \bar{y})\psi(\bar{x}, \bar{y})$, s.t. $I \models \varphi(\bar{a})$ for some assignment $\vec{a}$ on $\vec{x}$ and suppose that $I \not\models \exists \bar{y}\psi(\bar{a}, \bar{y})$. Then we extend I with the atoms in $At(\psi(\bar{a}, \bar{z}))$, where the elements of $\vec{z}$ are fresh labelled nulls. For the chase with s-t tgds, we stipulate that the new facts are added even if $I \models \exists \bar{y}\psi(\bar{a}, \bar{y})$ is already fulfilled. This kind of chase is referred to as *oblivious* [17] chase.

It is undecidable if the chase with a given set of target tgds terminates [6]. However, there are some broad classes of target tgds known to produce only finite chase sequences. Perhaps, the simplest such class is formed by full tgds, most general currently known classes being the super-weakly-acyclic tgds [19], or those in the hierarchy of inductively-restricted tgds [20]. In this paper, we only consider target tgds causing finite chase sequences, and corresponding *mappings with terminating chase property*.

Now suppose that Σ contains an egd $\varepsilon\colon \varphi(\bar{x}) \to x_i = x_j$, s.t. $I \models \varphi(\bar{a})$ for some assignment $\vec{a}$ on $\vec{x}$. This egd enforces the equality $a_i = a_j$. We thus choose a null a' among $\{a_i, a_j\}$ and replace *every occurrence* of a' in I by the other term; if $a_i, a_j \in Const(I)$ and $a_i \neq a_j$, the chase halts with *failure*. We write $chase(I, \Sigma)$ to denote the result of chasing I with the dependencies Σ. By slight abuse of notation, we shall also write $chase(I, \Sigma)$ to refer to the application of the chase procedure with the dependencies Σ to I.

Consider an arbitrary schema mapping $\Sigma = \Sigma_{st} \cup \Sigma_t$ where Σ_{st} is a set of source-to-target tgds and Σ_t is a set of target egds and tgds. Then the solution to a source instance I can be computed as follows: We start off with the instance $\langle I, \emptyset \rangle$, i.e., the source instance is I and the target instance is initially empty. Chasing $\langle I, \emptyset \rangle$ with Σ_{st} yields the instance $\langle I, J \rangle$, where J is called the *preuniversal instance*. This chase always succeeds since Σ_{st} contains no egds. Then J is chased with Σ_t. This chase may fail on an attempt to unify distinct constants. If the chase terminates and succeeds, we end up with $U = chase(J, \Sigma_t)$, which is referred to as the *canonical universal solution* of I w.r.t. Σ.

Equivalence of schema mappings. Different notions of equivalence of schema mappings have been recently proposed in [10].

DEFINITION 2.1. *[10] Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *and* $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ *be two schema mappings.*

$\mathcal{M}$ and $\mathcal{M}'$ are logically equivalent (denoted as $\mathcal{M} \equiv \mathcal{M}'$) if, for every source instance I and target instance J, the equivalence $\langle I, J\rangle \models \Sigma \Leftrightarrow \langle I, J\rangle \models \Sigma'$ holds. In this case, the equality $Sol(I, \mathcal{M}) = Sol(I, \mathcal{M}')$ holds for every source instance I.

$\mathcal{M}$ and $\mathcal{M}'$ are DE-equivalent (denoted as $\mathcal{M} \equiv_{DE} \mathcal{M}'$) if, for every source instance I, the universal solutions coincide, i.e.: the equality $UnivSol(I, \mathcal{M}) = UnivSol(I, \mathcal{M}')$ holds for every source instance I.

$\mathcal{M}$ and $\mathcal{M}'$ are CQ-equivalent (denoted as $\mathcal{M} \equiv_{CQ} \mathcal{M}'$) if, for every source instance I, either $Sol(I, \mathcal{M}) = \emptyset = Sol(I, \mathcal{M}')$ or $core(I, \mathcal{M}) = core(I, \mathcal{M}')$.

Remark. The original definition of $\mathcal{M} \equiv_{CQ} \mathcal{M}'$ is that, for every source instance I, any conjunctive query posed against the target schema yields the same certain answers for the mappings $\mathcal{M}_1$ and $\mathcal{M}_2$. The above characterization via the core was proved to be equivalent in [10].

By [4], we can use the chase to *decide logical implication* (and, hence, logical equivalence) of schema mappings consisting of embedded dependencies with terminating chase property.

LEMMA 2.1. *[4] Let Σ be the union of a set of egds and a set of tgds possessing terminating chase property. Moreover, let δ be either a tgd or an egd. Let $\varphi(\bar{x})$ denote the antecedent of δ and let T denote the database obtained by chasing $At(\varphi(\bar{x}))$ with Σ. The variables in $\vec{x}$ are considered as labelled nulls. Then $\Sigma \models \delta$ iff $T \models \delta$ holds.*

Turing machines. A *Turing machine* is a tuple $TM = (Q, A, \delta, s)$ given by a finite set of states Q, a finite set of symbols A called the alphabet, the transition function δ and an initial state s. For simplicity, Q and A are represented by integers $0, 1, \ldots$. The alphabet A contains at least the tape start symbol 1 (marking the left end of the tape) and the blank symbol 0. The set of states Q contains at least the initial state $s = 0$ and the halting state 1. The transition function δ is of the type $Q \times A \to Q \times A \times \{\leftarrow, \rightarrow\}$.

A *computation* C of *TM* is a sequence $(c_1, \ldots, c_n)$ of *configurations* c_t. For each configuration c_t, there is an associated state $q^{TM}(t)$, for each position l a tape symbol $a^{TM}(t, l)$ and a predicate indicating whether the cursor is currently at that location $c^{TM}(t, l)$.

W.l.o.g., we may restrict the instances of the Halting problem to instances of the following form: The Turing machine *TM* takes no input. Moreover, *TM* never returns to the initial state 0 and never moves the cursor off the left end of the tape.

3. CQ-EQUIVALENCE REVISITED

In [10], Fagin et al. showed that CQ-equivalence of schema mappings containing full s-t tgds and full target tgds is undecidable. This was accomplished by reduction from the undecidability of datalog equivalence [23].

THEOREM 3.1. [10] *CQ-equivalence is undecidable for schema mappings based on full s-t tgds and full target tgds.*

In essence, the idea is to identify datalog programs with full target tgds. In this section, we want to show that the same idea can be easily extended to prove further undecidability results concerning CQ-equivalence: On the one hand, we thus show the undecidability of CQ-equivalence of schema mappings containing s-t tgds and target egds. On the other hand, we also show the undecidability of various basic optimization tasks under CQ-equivalence.

THEOREM 3.2. *CQ-equivalence is undecidable for schema mappings based on s-t tgds and target egds.*

PROOF. (Sketch). Let $(\mathcal{M}_1, \mathcal{M}_2)$ be an arbitrary instance of CQ-equivalence for mappings based on full s-t tgds and full target tgds. From this we construct an equivalent instance $(\mathcal{M}'_1, \mathcal{M}'_2)$ of CQ-equivalence for mappings based on s-t tgds and target egds.

We replace each original full s-t tgd $\varphi(\bar{x}) \to Q(\bar{x})$ by

- $\varphi(\bar{x}) \to \exists y\, Q(\bar{x}, y, y)$

Furthermore, for each original target relation Q of arity n, and each possible combination of n attributes over (not necessarily distinct) original source relations $P_1, \ldots, P_n$, we add the following s-t tgd (thus materializing all relevant tuples in the target).

- $P_1(\ldots, x_1, \ldots) \wedge \ldots \wedge P_n(\ldots, x_n, \ldots) \to \exists y\, Q(\bar{x}, y, y')$

Finally, we replace each original full target tgd, which is of the form $R_1(\bar{x}_1) \wedge \ldots \wedge R_n(\bar{x}_n) \to R(\bar{x})$ by

- $R_1(\bar{x}_1, y_1, y_1) \wedge \ldots \wedge R_n(\bar{x}_n, y_n, y_n) \wedge R(\bar{x}, y, y') \to y = y'$

Intuitively, the two variables y, y' indicate through $y = y'$ that the specific tuple $\vec{x}$ is actually present. It is now easy to see that the two problem instances are equivalent. □

Theorems 3.1 and 3.2 hold even if restricted to single target dependencies. To show this, we make use of a result of Gottlob and Papadimitriou [14], who showed that for every datalog program, one can find an (essentially) equivalent datalog program which consists of a single rule (called sirup). Using this approach, it is easy to show that detecting a redundant target tgd is undecidable for CQ-equivalence.

THEOREM 3.3. *CQ-equivalence is undecidable for schema mappings based on full s-t tgds and a single full target tgd, or s-t tgds and a single target egd.*

PROOF. (Sketch). Putting together the undecidability proof in [10] (which generates one full target tgd per datalog rule) and the sirup construction from [14] (which produces a single full target tgd from any number of full target tgds), we end up with a single full target tgd. Using the technique introduced in Theorem 3.2, the result can be carried over to target egds. □

The main motivation for Fagin et al. to introduce the relaxed notions of equivalence [10] was the close connection between optimization and equivalence. Indeed, any *optimization* of schema mappings ultimately comes down to the replacement of a schema mapping by a *simpler* but *equivalent* one. A natural criterion for the simplification of schema mappings is the deletion of redundant dependencies. In [15], Gottlob et al. presented further criteria for simplifying a set of dependencies (with respect to logical equivalence) like subset-minimality and cardinality-minimality. In this paper, we thus want to study the following properties which are closely related to the optimization of schema mappings:

DEFINITION 3.1. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping and let $x \in \{DE, CQ\}$.*

1. *We call a dependency $\tau \in \Sigma$* redundant *w.r.t. x-equivalence, if $\Sigma \equiv_x \Sigma \setminus \{\tau\}$.*
2. *The set Σ is called* subset-minimal *w.r.t. x-equivalence, if there exists no proper subset $\Sigma' \subset \Sigma$, s.t. $\Sigma \equiv_x \Sigma'$ (i.e., Σ contains no redundant dependency).*
3. *The set Σ is called* cardinality-minimal *w.r.t. x-equivalence, if there does not exist a set Σ' of dependencies, s.t. $|\Sigma'| < |\Sigma|$ and $\Sigma \equiv_x \Sigma'$.* □

Below, we shall extend the above undecidability proofs to the first two properties in Definition 3.1, i.e.: detecting the redundancy of a concrete dependency and recognizing the subset-minimality of a set of dependencies w.r.t. CQ-equivalence.

THEOREM 3.4. *Detecting if a target dependency is redundant w.r.t. CQ-equivalence is undecidable for mappings based on full s-t tgds and two full target tgds, or s-t tgds and two target egds.*

PROOF. (Sketch). Let $(\mathcal{M}_1, \mathcal{M}_2)$ be an arbitrary instance of CQ-equivalence for mappings based on s-t tgds and a single full target tgd. We construct two instances $(\mathcal{M}, \sigma_1)$ and $(\mathcal{M}, \sigma_2)$ of detecting redundancy of σ_1 resp. σ_2 in $\mathcal{M}$ under CQ-equivalence. Then $\mathcal{M}_1 \equiv_{CQ} \mathcal{M}_2$ iff both of the following holds: σ_1 is redundant in $\mathcal{M}$ and σ_2 is redundant in $\mathcal{M}$.

To avoid interference, let $\mathcal{M}_1'$ (resp. $\mathcal{M}_2''$) be obtained by replacing all occurrences of target relation symbols R by new ones R' (resp. R''). Now let $\mathcal{M}$ consist of the s-t tgds of $\mathcal{M}_1'$ and $\mathcal{M}_2''$ as well as σ_1 and σ_2 given as follows: Construct the sirup σ_1 from

1. the single target tgd τ_1' of $\mathcal{M}_1'$ and
2. the single target tgd τ_2'' of $\mathcal{M}_2''$ and
3. target tgds $R'(\bar{x}) \rightarrow R(\bar{x})$ for each target relation symbol R' occurring in $\mathcal{M}_1'$.

Likewise, construct the sirup σ_2 based on R'' and $\mathcal{M}_2''$. The idea is that the transfer of all tuples from R' to R (via the tgds according to 3. above) is only necessary if it is not already covered by the transfer from R'' to R (and vice versa). It is now clear that $\mathcal{M}_1 \equiv_{CQ} \mathcal{M}_2$ iff σ_1 is redundant in $\mathcal{M}$ and σ_2 is redundant in $\mathcal{M}$. Again, this result can be easily carried over to target egds. □

Based on this technique, we can also show the undecidability of subset-minimality w.r.t. CQ-equivalence. The case for detecting redundant s-t tgds will be covered in Section 6.

THEOREM 3.5. *Subset-minimality w.r.t. CQ-equivalence is undecidable for schema mappings based on full s-t tgds and two full target tgds, or s-t tgds and two target egds.*

The proof technique based on the correspondence between datalog programs and full target tgds allowed us to give very short and straightforward proofs of several undecidability results in this section. However, we now reach at a limit of this technique. Note that in the proofs presented so far, we have very limited control over the precise form of the target dependencies. Hence, it is not clear, how this technique can be applied to the analysis of other properties of mappings (like cardinality-minimality proposed in Definition 3.1) or to restricted forms of mappings (like mappings whose target dependencies are functional dependencies or even key dependencies).

More importantly, we have only covered CQ-equivalence so far, while DE-equivalence has not been touched yet. As we have seen in Example 1.1, CQ- and DE-equivalence are distinct even for simple target constraints. So what is the situation for DE-equivalence? It does not seem as if the approach used in this section can be used to investigate DE-equivalence. Therefore, we present a completely different approach to these problems in the next sections.

4. DE-EQUIVALENCE

In this section, we consider schema mappings based on s-t tgds and either full target tgds or target egds. We will show that, in both cases, DE-equivalence is undecidable. This will be shown by reducing the Halting problem of Turing machines to the co-problem of DE-equivalence of such schema mappings. The principle of this reduction is very general and will be subsequently adapted to CQ- and DE-equivalence of various optimization problems (in Section 5) and to restricted forms of mappings (in Section 7). Therefore, we introduce a construction that works for both CQ- and DE-equivalence at the same time. The reduction is technically intricate, so we start by presenting the main ideas.

Principle. The general idea is to construct, for a Turing machine *TM*, two schema mappings $\mathcal{M}$ and $\mathcal{M}'$ such that

$$\textit{TM} \text{ halts} \quad \text{iff} \quad \mathcal{M} \not\equiv_{DE} \mathcal{M}'$$

Reformulated in terms of the behaviour of the schema mappings, this means that

- If *TM* halts, then for *at least one* source instance I, the difference between $\mathcal{M}$ and $\mathcal{M}'$ must become apparent (in terms of differing universal solutions).
- If *TM* does not halt, then for *all* source instances, $\mathcal{M}$ and $\mathcal{M}'$ must yield identical universal solutions. □

This construction is implemented through the following ideas:

Idea 1. For every Turing machine, there is a schema mapping using the target relations

$$\mathsf{state}(\cdot,\cdot) \quad \mathsf{tape}(\cdot,\cdot,\cdot) \quad \mathsf{cursor}(\cdot,\cdot,\cdot)$$

that can simulate a Turing machine computation of some specific length given an appropriate source instance. This can be done by just using s-t tgds and full target tgds (or target egds). We will call this a *simulation mapping*. □

Idea 2. There is an appropriate source instance to simulate all Turing machine computations of length n. It has the form

$$\mathsf{root}_S(1) \wedge \mathsf{chain}_S(1,2) \wedge \mathsf{chain}_S(2,3) \wedge \ldots \wedge \mathsf{chain}_S(n-1,n)$$

over the source schema

$$\mathsf{root}_S(\cdot) \quad \mathsf{chain}_S(\cdot,\cdot)$$

It will be called the *driving source instance* of length n. Through an infinite sequence of such instances, one can drive the simulation of all (arbitrarily long) terminating computations of *TM*. □

Idea 3. The reduction has to deal with *arbitrary* source instances. One can control the source instance in a way such that

- All unintended source instance properties that the Turing machine simulation cannot handle are *detected*.
- The remaining unintended properties can be *dealt with* in the Turing machine simulation.

This can be accomplished using just target egds through what we will call *source-guarding* dependencies. □

Idea 4. There is one dependency, which we call the *halting detection* dependency τ.

$$\mathsf{state}(\_, 1) \rightarrow \mathsf{halt}()$$

where 1 denotes the halting state. It is the only difference between the schema mappings $\mathcal{M}$ and $\mathcal{M}'$, i.e.

$$\mathcal{M}' = \mathcal{M} \cup \{\tau\}$$

This dependency τ only makes a difference iff *TM* halts. Notably, this construction works for both DE- and CQ-equivalence. □

4.1 Construction

We now outline the implementation of these general ideas by a schema mapping $\mathcal{M}_{TM} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ consisting of full s-t tgds Σ_{st} and target dependencies Σ_t containing full target tgds and target egds. $\mathcal{M}_{TM}$ will be referred to as the *simulation mapping*. Later we will show that the target tgds in this mapping can be replaced by target egds and vice versa.

We thus first define the source and target schemas $\mathbf{S}$ and $\mathbf{T}$, and the *driving source instances* I_n. Finally, the set of dependencies Σ will be presented.

Schemas. As discussed before, the source schema S contains atoms $\mathsf{chain}_S(\cdot,\cdot)$, intended to describe a linear order starting at $\mathsf{root}_S(\cdot)$.

$$\mathsf{root}_S(r) \quad \mathsf{chain}_S(px, x)$$

As a notational convention, we use the variable name px (and later also nx) to refer to the *previous* (resp. *next*) value relative to x in the chain structure.

In the target schema T, we hold a copy of this chain in

$$\mathsf{root}(r) \quad \mathsf{chain}(px, x)$$

We also have in the target schema three relations

$$\mathsf{state}(t, q) \quad \mathsf{tape}(t, l, a) \quad \mathsf{cursor}(t, l, c)$$

describing the state q, tape symbol a and cursor status c (i.e. if the cursor is present or not). The values for q, a and c are introduced by our dependencies. For easier presentation, we use constants (but this is not essential, as we will see later). The values for time t and tape location l will be taken from chain tuples in the source instance. Furthermore, we have the target relation

$$\mathsf{halt}()$$

to denote a halting Turing machine.

Source Instances. As illustrated before (in *Idea 2*), we need specific source instances to drive our simulation. We now define such source instances I_n, which drive the simulation of *TM*-computations of length n.

DEFINITION 4.1. *The* driving source instance I_n *of length* $n \geq 1$ *is given by the following relations over the domain* $\{1, \ldots, n\}$

$$\mathsf{root}_S^{I_n} = \{(1)\}, \quad \mathsf{chain}_S^{I_n} = \{(i-1, i) \mid 2 \leq i \leq n\}$$

Moreover, for $n = 0$*, we set* $I_n = \emptyset$*.*

In particular, if *TM* halts then one of these I_n will be the witness for differing universal solutions.

Source-to-target tgds. The first set of tgds are the *initialization tgds*. They establish the state, tape and cursor atoms for the initial time point. We refer to the tape starting symbol as 1 and to the blank symbol as 0. Moreover, the initial state is denoted by 0 and the halting state is denoted by 1.

- $\mathsf{root}_S(r) \rightarrow \mathsf{state}(r, 0)$
- $\mathsf{root}_S(r) \rightarrow \mathsf{tape}(r, r, 1)$
- $\mathsf{root}_S(r) \rightarrow \mathsf{cursor}(r, r, 1)$
- $\mathsf{root}_S(r) \wedge \mathsf{chain}_S(\_, x) \rightarrow \mathsf{tape}(r, x, 0)$
- $\mathsf{root}_S(r) \wedge \mathsf{chain}_S(\_, x) \rightarrow \mathsf{cursor}(r, x, 0)$

Here, the second and third tgd initialize the first tape position and the last two tgds initialize the subsequent tape positions. Additionally, we use the following *copy tgds* to transfer the chain from the source to the target.

- $\mathsf{root}_S(r) \rightarrow \mathsf{root}(r)$
- $\mathsf{chain}_S(px, x) \rightarrow \mathsf{chain}(px, x)$

Simulation full target tgds. We first construct a common left-hand side φ for all following dependencies. It matches the current state q, symbol a, and cursor position of the Turing machine:

$$\varphi := \mathsf{state}(t, q) \wedge \mathsf{tape}(t, l, a) \wedge \mathsf{cursor}(t, l, 1)$$

where 1 denotes that the cursor is present at that location.

We now construct *transition tgds* which compute the next state, symbol and cursor position. For each transition $\delta(q, a) = (q', a', d)$ of *TM*, we add the following dependencies to Σ_t.

- $\varphi \wedge \mathsf{chain}(t, t') \rightarrow \mathsf{state}(t', q')$
- $\varphi \wedge \mathsf{chain}(t, t') \rightarrow \mathsf{tape}(t', l, a')$

Depending on the cursor movement, we add one of the following two dependencies. If the transition encodes a cursor movement to the left ($d = \leftarrow$), add the first dependency, otherwise ($d = \rightarrow$) add the second one.

- $\varphi \wedge \mathsf{chain}(t, t') \wedge \mathsf{chain}(pl, l) \rightarrow \mathsf{cursor}(t', pl, 1)$
- $\varphi \wedge \mathsf{chain}(t, t') \wedge \mathsf{chain}(l, nl) \rightarrow \mathsf{cursor}(t', nl, 1)$

In addition to maintaining where the cursor is located, we also keep track of where the cursor is *not* located using *parity tgds*. The following tgd handles locations which are more than one move away from the current cursor position.

- $\mathsf{cursor}(t, pl, 0) \wedge \mathsf{cursor}(t, l, 0) \wedge \mathsf{cursor}(t, nl, 0) \wedge$
  $\mathsf{chain}(pl, l) \wedge \mathsf{chain}(l, nl) \wedge$
  $\mathsf{chain}(t, t') \rightarrow \mathsf{cursor}(t', l, 0)$

If the cursor is at most one move away from the current cursor position, we again add tgds for each transition $\delta(q, a) = (q', a', d)$ of *TM*. The first tgd below encodes that the cursor never stays at the same place. The second one is added if $d = \leftarrow$ and the third one if $d = \rightarrow$.

- $\varphi \wedge \mathsf{chain}(t, t') \rightarrow \mathsf{cursor}(t', l, 0)$
- $\varphi \wedge \mathsf{chain}(t, t') \wedge \mathsf{chain}(l, nl) \rightarrow \mathsf{cursor}(t', nl, 0)$
- $\varphi \wedge \mathsf{chain}(t, t') \wedge \mathsf{chain}(pl, l) \rightarrow \mathsf{cursor}(t', pl, 0)$

The reason for keeping track of locations where the cursor is not positioned is that for such locations, the tape symbol remains unchanged. This fact is expressed by the following *inertia tgd*.

- $\mathsf{tape}(t, l, a) \wedge \mathsf{cursor}(t, l, 0) \wedge$
  $\mathsf{chain}(t, t') \rightarrow \mathsf{tape}(t', l, a)$

Source-guarding dependencies. The dependencies defined so far are intended to work on driving source instances I_n. However, our reduction has to deal with arbitrary source instances. Below, we define additional *source-guarding dependencies* in order to either exclude certain unintended source instances or to control the effect of such source instances on the result of the chase.

The following egds check that there is at most one $\mathsf{root}(\_)$ and at most one predecessor and successor of a $\mathsf{chain}(\_,\_)$ atom. These egds indicate a violation of an intended structure by chase failure. We call them *no branching* egds.

- $\mathsf{root}(r) \wedge \mathsf{root}(r') \rightarrow r = r'$
- $\mathsf{chain}(px, x) \wedge \mathsf{chain}(px, x') \rightarrow x = x'$
- $\mathsf{chain}(px, x) \wedge \mathsf{chain}(px', x) \rightarrow px = px'$

Further deviations from the intended layout of source instances are not indicated by failure, but by *notification*. We implement this notification by indicating $\mathsf{halt}()$.

- $\mathsf{chain}_S(x, x) \rightarrow \mathsf{halt}()$
- $\mathsf{root}_S(r) \wedge \mathsf{chain}_S(\_, r) \rightarrow \mathsf{halt}()$

This completes the definition of the *simulation mapping* $\mathcal{M}_{TM} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ of a Turing machine *TM*.

4.2 Reduction

We are now going to show how the above construction indeed yields the undecidability of DE-equivalence. We thus first establish some key properties of the construction. After that, we illustrate a way to overcome the limited power of target dependencies to control the properties of the source instance. Finally, we give the undecidability proofs and extend them to important special cases.

DEFINITION 4.2. *Let J be a target instance containing the following atoms (where e_i denote pairwise distinct constants)*

$$\mathsf{root}(e_1) \wedge \mathsf{chain}(e_1, e_2) \wedge \mathsf{chain}(e_2, e_3) \ \ldots \wedge \mathsf{chain}(e_{n-1}, e_n),$$

where $e_1, \ldots, e_n$ are pairwise distinct constants. We say that the computation C of length n of Turing machine TM corresponds *to J, written $C \approx J$ iff the following holds for all $t, l \in \{1, \ldots, n\}$:*

$$q^{TM}(t) = q \Leftrightarrow \mathsf{state}^J(e_t, q) \qquad \neg c^{TM}(t, l) \Leftrightarrow \mathsf{cursor}^J(e_t, e_l, 0)$$
$$a^{TM}(t, l) = a \Leftrightarrow \mathsf{tape}^J(e_t, e_l, a) \quad c^{TM}(t, l) \Leftrightarrow \mathsf{cursor}^J(e_t, e_l, 1)$$

We now establish important properties of the construction from Section 4.1.

Let TM be a Turing machine and $\mathcal{M}_{TM} = (\mathbf{S}, \mathbf{T}, \Sigma)$ its simulation mapping. Let C denote a computation of TM of length n. Let I_n be the driving source instance and $J_n = chase(I_n, \Sigma)$.

LEMMA 4.1. *For each $t \in dom(I_n)$, there is exactly one atom $\mathsf{state}(t, \_)$ and for each $t \in dom(I_n)$ and $l \in dom(I_n)$, there is exactly one atom $\mathsf{tape}(t, l, \_)$ and $\mathsf{cursor}(t, l, \_)$ in J_n.*

LEMMA 4.2. *The chase $J_n = chase(I_n, \Sigma)$ will terminate and not trigger any source-guarding dependency. In particular, it will not fail and it will produce the same result as chasing I_n without the source-guarding dependencies.*

LEMMA 4.3. *The correspondence $C \approx J_n$ (according to Definition 4.2) holds.* □

The active chain. The source-guarding dependencies do not detect all unintended properties of a source instance. The instance I may contain no atom $\mathsf{root}(\_)$ or there may be several "chains", e.g. let $\mathsf{root}^I = \{(t_1)\}$ and

$$\mathsf{chain}^I = \{(t_1, t_2), (t_2, t_3), (t_3, t_4), (s_1, s_2), (s_2, s_3)\}$$

Then I contains two "chains" so to speak, namely t_1, t_2, t_3, t_4 and s_1, s_2, s_3. Clearly, only the first one is connected to the root. When not connected to the root, it is even possible that there is a cycle, e.g. let $\mathsf{root}^{I'} = \{(t_1)\}$ and

$$\mathsf{chain}^{I'} = \{(u_1, u_2), (u_2, u_3), (u_3, u_1)\}$$

For the chain connected to the root, this is ruled out by the source-guarding dependencies.

In the simulation mapping, the initial state is based on root and chain atoms connected to the root. Furthermore, dependencies in the simulation mapping only follow along the chain connected to the root. Therefore, while there may be several "chain"-like constructs, we are only interested in the one connected to the root. Below we define the *active chain* to formalize this intuition.

DEFINITION 4.3. *Let I be a source instance over the schema S of a simulation mapping. The active chain I^a is the minimal set of atoms that fulfills the following conditions.*

- $(\forall x)\ \mathsf{root}(x) \in I \Rightarrow \mathsf{root}(x) \in I^a$
- $(\forall px, x)\ \mathsf{root}(px) \in I \wedge \mathsf{chain}(px, x) \in I \Rightarrow \mathsf{chain}(px, x) \in I^a$
- $(\forall px, x, nx)\ \mathsf{chain}(px, x) \in I^a \wedge \mathsf{chain}(x, nx) \in I \Rightarrow \mathsf{chain}(x, nx) \in I^a$

We now establish crucial properties of the active chain.

Let TM be a Turing machine and $\mathcal{M}_{TM} = (\mathbf{S}, \mathbf{T}, \Sigma)$ its simulation mapping. For an arbitrary source instance I, let I^a be the active chain. Let $J = chase(I, \Sigma)$ and $J^a = chase(I^a, \Sigma)$. Suppose that the chase of I with Σ does not trigger any source-guarding dependencies.

LEMMA 4.4. *The active chain I^a is isomorphic to the driving source instance I_n of length $n = |I^a|$.*

LEMMA 4.5. *All atoms derived by the target chase of J are already generated by the target chase of J^a.* □

THEOREM 4.6. *DE- and CQ-equivalence are undecidable for mappings based on full s-t tgds, full target tgds and target egds.*

PROOF. We proceed by reducing the Halting problem to the co-problem of DE- resp. CQ-equivalence. Let *TM* be an arbitrary instance of Halting. Then we define the two schema mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma \cup \{\tau\})$, where $\mathcal{M} = \mathcal{M}_{TM}$ is the simulation mapping for *TM* and τ is defined as $\mathsf{state}(\_, 1) \rightarrow \mathsf{halt}()$. It remains to show that *TM* halts iff $\mathcal{M} \not\equiv_{CQ} \mathcal{M}'$ resp. $\mathcal{M} \not\equiv_{DE} \mathcal{M}'$.

The proof is given in three claims:

CLAIM 1. If *TM* halts then $\mathcal{M} \not\equiv_{CQ} \mathcal{M}'$
CLAIM 2. If $\mathcal{M} \not\equiv_{CQ} \mathcal{M}'$ then *TM* halts
CLAIM 3. $\mathcal{M} \not\equiv_{DE} \mathcal{M}'$ implies $\mathcal{M} \not\equiv_{CQ} \mathcal{M}'$

Moreover, for arbitrary schema mappings we have the reverse direction of Claim 3, $\mathcal{M} \not\equiv_{CQ} \mathcal{M}'$ implies $\mathcal{M} \not\equiv_{DE} \mathcal{M}'$. With that, it is clear that *TM* halts iff $\mathcal{M} \not\equiv_{CQ} \mathcal{M}'$ resp. $\mathcal{M} \not\equiv_{DE} \mathcal{M}'$. □

We now show that undecidability even holds if the target dependencies are restricted to egds only or to full tgds only. Note that for full target tgds only, it suffices to consider full s-t tgds, but for target egds only, we need to consider non-full s-t tgds.

THEOREM 4.7. *DE- and CQ-equivalence are undecidable for schema mappings based on s-t tgds and target egds.*

PROOF. (Sketch). We start with the simulation mapping as introduced for Theorem 4.6. This mapping uses only full tgds and egds. Therefore we encode the full tgds into egds using the technique introduced in Theorem 3.2. □

THEOREM 4.8. *DE- and CQ-equivalence are undecidable for schema mappings based on full s-t tgds and full target tgds.*

PROOF. (Sketch). We again start with the simulation mapping as introduced for Theorem 4.6. We eliminate all egds from the mapping by adapting a technique used in [7]. This is done by introducing a relation symbol $E(\cdot, \cdot)$ denoting equality. □

5. UNDECIDABILITY OF OPTIMIZATION

We now extend the above undecidability results to several natural optimization tasks. More precisely, we show that all properties presented in Definition 3.1 are undecidable both for DE- and CQ-equivalence. We only give proofs for the case of mappings where all target dependencies are full tgds. By the proof idea of Theorem 3.2, we can easily replace the full tgds by egds.

The undecidability of checking if a specific dependency is redundant follows directly from the proof of Theorems 4.7 and 4.8.

COROLLARY 5.1. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a a schema mapping with $\Sigma = \Sigma_{st} \cup \Sigma_t$ based on s-t tgds and target egds, or full s-t tgds and full target tgds and let $\tau \in \Sigma$. Then it is undecidable if τ is redundant w.r.t. DE- resp. CQ-equivalence.*

PROOF. Recall the simulation mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ with $\Sigma' = \Sigma \cup \{\tau\}$ introduced for Theorem 4.6. Clearly, τ is redundant in $\mathcal{M}'$, iff $\mathcal{M} \equiv_{CQ} \mathcal{M}'$ resp. $\mathcal{M} \equiv_{DE} \mathcal{M}'$. By Theorems 4.7 and 4.8, deciding these equivalences would come down to deciding the Halting problem. □

We now show that undecidability also holds for subset-minimality for both DE- and CQ-equivalence. In the preceding corollary, the main idea was that a specific dependency τ is non-redundant iff *TM* halts. Therefore for subset-minimality, it suffices to make sure that τ is the *only* dependency that might be redundant.

THEOREM 5.2. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a a schema mapping with $\Sigma = \Sigma_{st} \cup \Sigma_t$ based on s-t tgds and target egds, or full s-t tgds and full target tgds. Then it is undecidable if Σ is subset-minimal w.r.t. DE- resp. CQ-equivalence.*

PROOF. (Sketch). The mapping $\mathcal{M}'$ from Theorem 4.6 can be transformed into a mapping $\mathcal{M}''$ where apart from the halting detection dependency τ, no other dependency can be redundant. This non-redundancy of any other dependency σ is accomplished by adding appropriate s-t tgds which cause σ to "fire" when chasing an appropriately chosen source instance with $\mathcal{M}''$. □

For cardinality-minimality we have to combine the proof ideas of the previous undecidability proofs with the sirup construction [14], which already played an important role in Section 3.

THEOREM 5.3. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a a schema mapping with $\Sigma = \Sigma_{st} \cup \Sigma_t$ based on s-t tgds and target egds, or full s-t tgds and full target tgds. Then it is undecidable if Σ_t is cardinality-minimal w.r.t. DE- resp. CQ-equivalence.*

PROOF. (Sketch). We start with our simulation mapping $\mathcal{M}$ from Theorem 4.6. The work of the s-t tgds in Section 4.1 and the setup of relations needed for the sirup construction can be done by a set of s-t tgds that cannot be further reduced. As in the proof of Theorem 4.8, we can replace the target egds by full target tgds using an equality relation. Based on the sirup construction of [14] that we already used in the proof of Theorem 3.3, we know how to encode all our full target tgds by a single full target tgd σ.

In summary, we can transform the simulation mapping $\mathcal{M}$ from Section 4 into a mapping $\mathcal{M}'$ consisting of a set of s-t tgds Σ_{st} that cannot be further reduced and a single full target tgd σ. Now let $\mathcal{M}'' = \mathcal{M}' \cup \{\tau\}$, where τ is the halting detection dependency from Section 4. Then $\mathcal{M}''$ is cardinality-minimal iff τ is not redundant in $\mathcal{M}''$ iff *TM* halts. □

In [15], Gottlob et al. also looked at other optimization criteria besides subset- and cardinality-minimality. One such criterion is *antecedent-minimality*, which we want to apply to the target egds here, i.e.: Given a mapping $\Sigma_{st} \cup \Sigma_t$, where Σ_t consists of egds only, is the total number of atoms in all antecedents in Σ_t minimal?

THEOREM 5.4. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a a mapping with $\Sigma = \Sigma_{st} \cup \Sigma_t$ of s-t tgds and target egds. Then it is undecidable if for all Σ'_t, s.t. $\Sigma_{st} \cup \Sigma_t \equiv_{CQ} \Sigma_{st} \cup \Sigma'_t$ (resp. $\Sigma_{st} \cup \Sigma_t \equiv_{DE} \Sigma_{st} \cup \Sigma'_t$), the total number of atoms in the antecedents in Σ_t is less than or equal to the total number of atoms in the antecedents in Σ'_t. That is, Σ_t is antecedent-minimal w.r.t. DE- resp. CQ-equivalence.*

PROOF. (Sketch). For this, we exploit a result shown later for Theorem 7.1: Our simulation mapping $\mathcal{M}$ from Theorem 4.6 can be transformed into a mapping $\mathcal{M}^*$ which uses at most one key dependency per target relation. Then $\mathcal{M}^*$ is clearly cardinality-minimal, since there is only one dependency enforcing equalities in each relation, and this one cannot be left out (without violating equivalence). □

6. OPTIMIZATION OF S-T TGDS

The undecidability results proved in the previous sections are mainly due to the target dependencies. That is, DE- and CQ-equivalence of two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ remain undecidable even if $\mathcal{M}_1$ and $\mathcal{M}_2$ coincide on the s-t tgds (and they only differ on the target dependencies). Likewise, several optimization tasks of the target dependencies are undecidable even if the s-t tgds are kept fixed. We now shift our focus to the s-t-tgds and investigate two main questions: First, do the relaxed notions of equivalence of two schema mappings become decidable if the schema mappings coincide on the target dependencies? Second, do the relaxed notions of equivalence provide additional potential (compared with logical equivalence) of optimization of the s-t tgds in the presence of target dependencies? Below, we shall give a positive answer to the first question and a negative answer to the second one.

The main technical tool for deriving these results will be an adaptation and extension of the PROPAGATE procedure from [15]. In [15], this procedure was used to "propagate" the effect of the target egds into the s-t tgds. It thus played an important role in the computation of a normal form of s-t tgds in the presence of target egds. In Figure 1, we present the extended version of this procedure, called PROPAGATE^E. Its input is (1) a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\Sigma = \Sigma_{st} \cup \Sigma_t$ consists of s-t tgds and target egds and tgds, and (2) a conjunction of atoms $\varphi(\bar{x})$, where $\vec{x}$ denotes the variables in this conjunction. By chasing $At(\varphi(\bar{x}))$ with Σ, PROPAGATE^E derives a set of source egds $\Delta_s^{\Sigma}(\varphi(\bar{x}))$ and a set $\Delta_{st}^{\Sigma}(\varphi(\bar{x}))$ consisting of a single s-t tgd. The following example will help to illustrate the work of the PROPAGATE^E procedure.

EXAMPLE 6.1. Consider the schema mapping $\Sigma = \{\tau_1, \tau_2, \varepsilon\}$ consisting of a single s-t tgd, one target tgd, and one target egd:

$\tau_1 \colon S(x_1, x_2) \rightarrow (\exists y) P(x_1, y), Q(y, x_2)$
$\tau_2 \colon P(x_1, x_2) \rightarrow (\exists z) R(x_2, x_1, z)$
$\varepsilon \colon R(x, v, z) \wedge Q(x, w) \rightarrow v = w$

Suppose that we apply the PROPAGATE^E procedure to the CQ $\varphi(\bar{x}) = S(x_1, x_2)$, i.e., $\varphi(\bar{x})$ is the antecedent of the s-t tgd τ_1. Then the procedure starts with source database $I = \{S(x_1, x_2)\}$. The chase of I with Σ takes three steps. The first two derive new facts from I: $J_1 = \{P(x_1, y), Q(y, x_2)\}$, $J_2 = J_1 \cup \{R(y, x_1, z)\}$. The last chase step enforces the egd ε by unifying x_1 and x_2, i.e., for substitution $\lambda = \{x_1 \leftarrow x, x_2 \leftarrow x\}$, we have $J_3 = J_2\lambda = \{P(x, y), R(y, x, z), Q(y, x)\}$.

In total, the procedure derives a source egd $\sigma \colon S(x_1, x_2) \rightarrow x_1 = x_2$, and the s-t tgd $\tau'_1 \colon S(x, x) \rightarrow (\exists y, z) P(x, y) \wedge R(y, x, z) \wedge Q(y, x)$. It can be easily checked that $\Sigma \equiv \Sigma'$ with $\Sigma' = \{\sigma, \tau'_1, \tau_2, \varepsilon\}$, i.e., it is correct (up to logical equivalence) to replace τ_1 in Σ by the source egd and the s-t tgd resulting from PROPAGATE^E. In particular, the source egd σ is logically implied by Σ, i.e.: the chase with Σ fails on every source instance containing a fact $S(c_1, c_2)$ where c_1, c_2 are distinct constants. □

Generalizing Example 6.1, we can prove the following properties of the PROPAGATE^E procedure: (1) PROPAGATE^E is *sound*, i.e., it only derives dependencies which are logically implied by Σ. (2) If PROPAGATE^E is called with the antecedent $\varphi(\bar{x})$ of some dependency $\tau \in \Sigma$, then τ may be replaced in Σ by the source egds and s-t tgds resulting from the PROPAGATE^E procedure. Finally, we also show that (3) PROPAGATE^E does not distinguish between CQ-equivalent mappings. Formally, we get the following lemmas.

LEMMA 6.1. *Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ and let $\varphi(\bar{x})$ be a an arbitrary CQ. Then $\Sigma \models \Delta_s^{\Sigma}(\varphi(\bar{x})) \cup \Delta_{st}^{\Sigma}(\varphi(\bar{x}))$.*

Procedure PROPAGATE^E
Input: Mapping $\Sigma = \Sigma_{st} \cup \Sigma_t$, conjunction $\varphi(\bar{x})$
Output: Sets of dependencies $\Delta_s^{\Sigma}(\varphi(\bar{x}))$ and $\Delta_{st}^{\Sigma}(\varphi(\bar{x}))$

/* 1. chase with $\Sigma = \Sigma_{st} \cup \Sigma_t$ */
$I := At(\varphi(\bar{x}))$;
$J := chase(I, \Sigma)$;

/* 2. compute s-t tgd τ */
let $J = J_S \cup J_T$, s.t. J_S is an instance over $\mathbf{S}$
and J_T is an instance over $\mathbf{T}$;
let $J^* = core(J_T)$, where the variables that occur
in J_S are considered as constants.
$\tau := \big(\bigwedge_{A \in J_S} A\big) \rightarrow (\exists \bar{y}\,) \bigwedge_{B \in J^*} B$;
$\Delta_{st} := \{\tau\}$;

/* 3. compute source egds */
$\Delta_s := \emptyset$;
Compute a substitution λ s.t. $At(\varphi(\bar{x}\lambda)) = J_S$;
for each pair of variables $x_j, x_k \in \bar{x}$ **do**
if $x_j\lambda = x_k\lambda$ **then**
$\Delta_s := \Delta_s \cup \{\varphi(\bar{x}\,) \rightarrow x_j = x_k\}$;

/* 4. output result */
return (Δ_s, Δ_{st});

Figure 1: Extended Propagate Procedure.

LEMMA 6.2. *Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\sigma \in \Sigma_{st}$, s.t. $\varphi(\bar{x})$ is the antecedent of σ and let $\Sigma' = (\Sigma \setminus \{\sigma\}) \cup \Delta_s^{\Sigma}(\varphi(\bar{x})) \cup \Delta_{st}^{\Sigma}(\varphi(\bar{x}))$. Then $\Sigma' \models \Sigma$ holds.*

LEMMA 6.3. *Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\Upsilon = \Upsilon_{st} \cup \Upsilon_t$ be schema mappings with $\Sigma \equiv_{CQ} \Upsilon$. Moreover, for an arbitrary conjunction $\varphi(\bar{x})$, let $\Delta^{\Sigma} = \Delta_s^{\Sigma}(\varphi(\bar{x})) \cup \Delta_{st}^{\Sigma}(\varphi(\bar{x}))$ and $\Delta^{\Upsilon} = \Delta_s^{\Upsilon}(\varphi(\bar{x})) \cup \Delta_{st}^{\Upsilon}(\varphi(\bar{x}))$, respectively, denote the output of the* PROPAGATE^E *procedure. Then $\Delta^{\Sigma} \equiv \Delta^{\Upsilon}$ holds.*

We now combine these three lemmas to show that if two mappings are CQ-equivalent, then their difference can be reduced to the target dependencies (while the s-t tgds can be replaced by common source egds and s-t tgds). Since DE-equivalence implies CQ-equivalence, this clearly holds for DE-equivalent mappings as well.

THEOREM 6.4. *Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\Upsilon = \Upsilon_{st} \cup \Upsilon_t$ with $\Sigma \equiv_{CQ} \Upsilon$. Then there exist a set Σ_s^* of source egds and Σ_{st}^* of s-t tgds, s.t. $\Sigma \equiv \Sigma_s^* \cup \Sigma_{st}^* \cup \Sigma_t$ and $\Upsilon \equiv \Sigma_s^* \cup \Sigma_{st}^* \cup \Upsilon_t$.*

PROOF. The idea is to call PROPAGATE^E with every antecedent occurring in $\Sigma_{st} \cup \Upsilon_{st}$ and to take Σ_s^* (resp. Σ_{st}^*) as the set of all source egds (resp. all s-t tgds) produced by these procedure calls. Formally, we set $\Phi = \{\varphi(x) \mid \varphi(x)$ is the antecedent of some $\sigma \in \Sigma_{st}\} \cup \{\varphi(x) \mid \varphi(x)$ is the antecedent of some $\sigma \in \Upsilon_{st}\}$ and $\Sigma_s^* = \bigcup_{\varphi(x) \in \Phi} \Delta_s^{\Sigma}(\varphi(x))$ and $\Sigma_{st}^* = \bigcup_{\varphi(x) \in \Phi} \Delta_{st}^{\Sigma}(\varphi(x))$. The (logical) equivalences $\Sigma \equiv \Sigma_s^* \cup \Sigma_{st}^* \cup \Sigma_t$. and $\Upsilon \equiv \Sigma_s^* \cup \Sigma_{st}^* \cup \Upsilon_t$ are shown by induction on $|\Phi|$ and using the Lemmas 6.1 – 6.3. □

With this theorem at hand, we can now settle the question of the decidability of DE- and CQ-equivalence of schema mappings with logically equivalent (and, in particular, with identical) target dependencies. To this end, we first prove the following lemma.

LEMMA 6.5. *Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\Upsilon = \Upsilon_{st} \cup \Upsilon_t$, such that $\Sigma \equiv_{CQ} \Upsilon$ and $\Sigma_t \equiv \Upsilon_t$ holds. Then $\Sigma \equiv \Upsilon$.*

PROOF. By Theorem 6.4, there exist a set of source egds ${\Sigma_t}^*$ and a set of s-t tgds Σ_{st}^* such that $\Sigma \equiv \Sigma_s^* \cup \Sigma_{st}^* \cup \Sigma_t$ and $\Upsilon \equiv \Sigma_s^* \cup \Sigma_{st}^* \cup \Upsilon_t$. The claim follows from $\Sigma_t \equiv \Upsilon_t$. □

THEOREM 6.6. *Suppose that the problems of DE- and CQ-equivalence are restricted to pairs of schema mappings $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\Upsilon = \Upsilon_{st} \cup \Upsilon_t$, s.t. Σ_t and Υ_t are* logically equivalent. *With this restriction, the DE- and CQ-equivalence problems $\Sigma \equiv_{DE} \Upsilon$ and $\Sigma \equiv_{CQ} \Upsilon$, respectively, are decidable.*

PROOF. By the condition $\Sigma_t \equiv \Upsilon_t$ and by Lemma 6.5, $\Sigma \equiv \Upsilon$ holds iff $\Sigma \equiv_{CQ} \Upsilon$ holds. Hence, also $\Sigma \equiv_{DE} \Upsilon$ coincides with logical equivalence. □

In other words, the relaxed notions of equivalence are decidable on schema mappings if only the s-t tgds are allowed to vary. This is in sharp contrast to the situation where only the target dependencies vary, see Section 4. This naturally raises the question if the decidability result of Theorem 6.6 allows us to exploit additional possibilities (compared with logical equivalence) of optimizing the s-t tgds in the presence of target dependencies. The following example illustrates the simplifications achievable with logical equivalence.

EXAMPLE 6.2. Consider the mapping $\Sigma = \{\tau_1, \tau_2\}$, where τ_1 (resp. τ_2) is the following s-t tgd (resp. target egd):

$\tau_1\colon S(x_1, x_2) \wedge S(x_1, x_3) \rightarrow P(x_2, y_1) \wedge Q(y_2, x_3) \wedge Q(y_3, x_3)$
$\tau_2\colon P(x_1, v) \wedge Q(w, x_2) \rightarrow v = w$

To simplify the notation, we have omitted the existential quantification (which has to be applied to the variables occurring in the conclusion only). Under logical equivalence, Σ can be simplified to $\Sigma' = \{\tau_1', \tau_2\}$ with $\tau_1'\colon S(x_1, x_2) \rightarrow P(x_2, y_1) \wedge Q(y_2, x_2)$.

Using the dependency implication test from Lemma 2.1, one can easily verify that $\Sigma' \models \tau_1$ and $\Sigma \models \tau_1'$, that is, $\Sigma \equiv \Sigma'$ holds. □

We now show for a broad class of optimization problems on s-t tgds that the relaxed notions of equivalence lead to exactly the same notion of optimality as logical equivalence. Negatively, this means that the relaxed notions of equivalence do not give us additional power. Positively, this means that for optimizing the s-t tgds one can interchangeably use algorithms for any of these notions of equivalence. Below, we carry over the notions of optimality from Definition 3.1 to s-t tgds and generalize these notions to arbitrary, real-valued target functions. W.l.o.g., we restrict ourselves to minimization problems. For the sake of a uniform notation, we denote logical equivalence by $\equiv_{log}$ rather than $\equiv$.

DEFINITION 6.1. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ with $\Sigma = \Sigma_{st} \cup \Sigma_t$ be a schema mapping, let $x \in \{log, DE, CQ\}$ and let F be a function that assigns a real number to every set of s-t tgds.*

Then the set of s-t tgds Σ_{st} is called F-optimal *w.r.t. x-equivalence, if there does not exist a set of s-t tgds Σ_{st}', s.t. $F(\Sigma_{st}') < F(\Sigma_{st})$ and $\Sigma_{st} \cup \Sigma_t \equiv_x \Sigma_{st}' \cup \Sigma_t$.*

Formally, we show below that, for any real-valued function F, the F-optimality w.r.t. DE- or CQ-equivalence coincides with the F-optimality w.r.t. logical equivalence.

THEOREM 6.7. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ with $\Sigma = \Sigma_{st} \cup \Sigma_t$ be a schema mapping and let F be a function that assigns a real number to every set of s-t tgds. Then Σ_{st} is F-optimal w.r.t. logical equivalence iff it is F-optimal w.r.t. DE-equivalence iff it is F-optimal w.r.t. CQ-equivalence.*

PROOF. Logical equivalence entails DE-equivalence, which entails CQ equivalence [10]. Hence, it suffices to show that if Σ_{st} is F-optimal w.r.t. logical equivalence then it is also F-optimal w.r.t. CQ-equivalence. Suppose to the contrary that Σ_{st} is F-optimal w.r.t. logical equivalence but not w.r.t. CQ-equivalence. Then there exists Σ_{st}', s.t. $F(\Sigma_{st}') < F(\Sigma_{st})$ and $\Sigma_{st} \cup \Sigma_t \equiv_{CQ} \Sigma_{st}' \cup \Sigma_t$. By Lemma 6.5, then also $\Sigma_{st} \cup \Sigma_t \equiv \Sigma_{st}' \cup \Sigma_t$ holds, which contradicts the assumption that Σ_{st} is F-optimal w.r.t. logical equivalence. □

7. RESTRICTED CLASSES OF MAPPINGS

In this section, we consider schema mappings with specific target dependencies: functional dependencies, key dependencies and inclusion dependencies. We first show that the CQ equivalence remains undecidable for such mappings. DE-equivalence, however, becomes decidable for the class of mappings which includes a fairly broad class of common database constraints, namely functional and inclusion dependencies possessing the terminating chase property.

7.1 Target KDs under CQ-Equivalence

We now show undecidability of CQ-equivalence for mappings using at most one key dependency per target relation. Essentially, this means simulating a Turing machine using such mappings.

For the simulation, we need to compute the next state of the Turing machine given its current state. This was originally done using a target tgd of the form

- $\mathsf{state}(t, q) \wedge \ldots \wedge \mathsf{chain}(t, t') \rightarrow \mathsf{state}(t', q')$ (1)

Clearly, this cannot be done using key dependencies.

Idea 1. Given the current state q at time t and a number of other values (like tape symbol), it is possible to express the next state q' as a key dependency using the relation $\mathsf{next\text{-}state}(t, \ldots, q, q')$.

For each *possible* transition, we use s-t tgds to create next-state atoms containing only constants. E.g., suppose there is a transition from state 3 to state 4 given specific conditions $\vec{c}$. Then we need to add atoms for all possible times $0, 1, \ldots, n$:

$$\begin{gathered}\mathsf{next\text{-}state}(0, \bar{c}, 3, 4)\\ \mathsf{next\text{-}state}(1, \bar{c}, 3, 4)\\ \ldots\\ \mathsf{next\text{-}state}(n, \bar{c}, 3, 4)\end{gathered}$$

Now we can use a key dependency to do lookups into that relation.

- $\mathsf{next\text{-}state}(\bar{x}, q, q') \wedge \mathsf{next\text{-}state}(\bar{x}, q, q'') \rightarrow q' = q''$

E.g. at time 7, if we want to retrieve the resulting state q' of an *actual* transition from state 3 and specific conditions $\vec{c}$, we can do this via an atom $\mathsf{next\text{-}state}(7, \bar{c}, 3, q')$ in the target instance: Our key dependency enforces the instantiation of q' to 4.

The construction of all of the needed lookup tables (for state, symbol, cursor) can be done using similar s-t tgds. □

Given these relations, we need to describe a full computation. So we use the following s-t tgd to establish lookups into next-state.

- $\mathsf{chain}_S(t, \_) \rightarrow (\exists q, q')\ \mathsf{next\text{-}state}(t, \ldots, q, q')$ (2)

So, for a computation of length n, we need to make these lookups:

$$\begin{gathered}\mathsf{next\text{-}state}(0, \ldots, q_0, q_0')\\ \mathsf{next\text{-}state}(1, \ldots, q_1, q_1')\\ \ldots\\ \mathsf{next\text{-}state}(n, \ldots, q_n, q_n')\end{gathered}$$

However, there is a problem, since the result of the first lookup q_0' is not propagated to the argument q_1 of the second lookup.

What is clear is that we cannot use s-t tgds to guarantee $q_i' = q_{i+1}$ using existential variables. After all, the length of the computation is unknown during the design of our schema mapping.

Idea 2. It is possible to connect variables along an indefinite number of atoms using key dependencies.

In fact, we want that the states $q_0, q_0', q_1, q_1', \ldots$ from next-state are exactly the unique states at times $0, 1, \ldots$ given by the state relation. We therefore extend s-t tgd (2) to additionally store which states correspond to which time points.

- $\mathsf{chain}_S(t, t') \rightarrow (\exists q, q')\ \mathsf{next\text{-}state}(t, \ldots, q, q') \wedge \mathsf{state}(t, q) \wedge \mathsf{state}(t', q')$ (3)

We then use a single key dependency to actually ensure that for each time point t, there is a specific single state q.

- $\mathsf{state}(t, q) \wedge \mathsf{state}(t, q') \rightarrow q = q'$

E.g., at times 0 and 1, we have the following tuples generated by our s-t tgd

$$\begin{aligned}&\mathsf{next\text{-}state}(0, \ldots, q_0, q_0') \wedge \mathsf{state}(0, q_0) \wedge \mathsf{state}(1, q_0') \wedge\\ &\mathsf{next\text{-}state}(1, \ldots, q_1, q_1') \wedge \mathsf{state}(1, q_1) \wedge \mathsf{state}(2, q_1')\end{aligned}$$

So since the target instance contains the atoms $\mathsf{state}(1, q_0')$ and $\mathsf{state}(1, q_1)$, the key dependency enforces the equality $q_0' = q_1$. This technique is used for state in one dimension (time) and for tape and cursor in two dimensions (time and tape location). □

So far, we can simulate Turing machines, but we still need a mechanism to detect halting. Similar to the construction in Section 4, we would like to use the tgd $\mathsf{state}(\_, 1) \wedge \mathsf{halt}(h, h') \rightarrow h = h'$ for this purpose. However, it is not a key dependency.

Idea 3. To effectively detect the halting state, we introduce a relation detect to associate every state q with some arbitrarily chosen constant 0. We realize this by extending s-t tgd (3) as follows

- $\mathsf{chain}_S(t, t') \rightarrow (\exists q, q')\ \mathsf{next\text{-}state}(t, \ldots, q, q') \wedge \mathsf{state}(t, q) \wedge \mathsf{state}(t', q') \wedge \mathsf{detect}(q, 0)$ (4)

Furthermore, we add an atom $\mathsf{halt}(x)$ and a detect atom which associates the halting state (state 1) with the variable x.

- $\mathsf{root}_S(\_) \rightarrow \mathsf{detect}(1, x) \wedge \mathsf{halt}(x)$

For halting detection, we add the following key dependency

- $\mathsf{detect}(q, y) \wedge \mathsf{detect}(q, y') \rightarrow y = y' \quad [\tau]$

Now if some q equals the halting state (state 1), the variable x will be set to 0 using τ. So let $\mathcal{M}$ be defined by the dependencies sketched above and assume an appropriate source instance I. Then the the core of I under $\mathcal{M}$ contains $\mathsf{halt}(0)$ iff *TM* halts, and it contains $\mathsf{halt}(v)$ for a labelled null v iff *TM* does not halt. □

THEOREM 7.1. *CQ-equivalence is undecidable for schema mappings based on s-t tgds and at most one key dependency per target relation.*

At this point, it is important to observe that the construction in the preceding theorem does not work for DE-equivalence. This is the case because of the form of the halting detection dependency τ.

In particular, assume that *TM* does not halt. Then the labelled null v in any of the atoms $\mathsf{state}(\_, v)$ generated by the s-t tgd in (4) will never be instantiated to 1. Hence, we know that τ never fires. Let $\mathcal{M}' = \mathcal{M} \cup \{\tau\}$ and let I_n (with $n > 0$) be a driving source instance. Then the chase of I_n with $\mathcal{M}$ and also with $\mathcal{M}'$ succeeds and $core(I_n, \mathcal{M}) = core(I_n, \mathcal{M}')$. Now let J be an arbitrary universal solution of I_n under $\mathcal{M}$. Then we know that $\mathsf{detect}(0, 0)$ must be contained in J (since state 0 is the initial state). So let us extend J to $J' = J \cup \{\mathsf{detect}(0, u)\}$ for a fresh labelled null u. This new atom will not be present in the core, so J' does not contradict $\mathcal{M} \equiv_{CQ} \mathcal{M}'$. However, J' violates τ, therefore J' is not a solution under $\mathcal{M}'$. So $\mathcal{M} \not\equiv_{DE} \mathcal{M}'$ since $J' \in UnivSol(I, \mathcal{M})$ but $J' \notin Sol(I, \mathcal{M}')$ and, therefore $J' \notin UnivSol(I, \mathcal{M}')$.

In contrast, if we do not require the halting detection dependency to be a key dependency, then the construction is applicable to both CQ- and DE-equivalence. This was indeed the case for Theorem 5.4, which exploits the preceding construction, but with a non-key dependency for halting detection.

7.2 Target IDs and FDs under DE-Equivalence

The disparity between DE- and CQ-equivalence observed in the previous section will become yet deeper in this section: We show that DE-equivalence is decidable for mappings whose target dependencies even belong to a bigger class than just key dependencies. The idea of the equivalence test is rather simple: given two mappings, we identify dependencies in one mapping which are not logically implied by the other mapping. For DE-equivalence, it is sufficient that no such "differing" dependency ever fires in any possible chase sequence (see Theorem 7.3). Moreover, we will show that for a certain class of target dependencies, this condition is also necessary for DE-equivalence.

DEFINITION 7.1. *A query φ is said to be satisfiable in a mapping Σ if there exists a source instance I such that the chase of I with Σ terminates and succeeds, and $chase(I, \Sigma) \models \varphi$ holds.*

DEFINITION 7.2. *Let Σ and Σ' be schema mappings. Dependency $\tau \in \Sigma \cup \Sigma'$ is said to be "differing", if it is not implied by one of the mappings, i.e., either $\Sigma \not\models \tau$ or $\Sigma' \not\models \tau$ holds.*

DEFINITION 7.3 (AUC). *We say that a pair of mappings Σ, Σ' meets the* antecedent unsatisfiability condition (AUC)*, if no differing dependency τ in any of the two mappings has an antecedent satisfiable in the mapping which contains τ.*

As it was shown by Theorem 6.4, for any two mappings $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\Sigma' = \Sigma'_t \cup \Sigma'_{st}$, whenever $\Sigma \equiv_{CQ} \Sigma'$ holds, there exist sets Σ^*_s of source egds and Σ^*_{st} of s-t tgds, such that $\Sigma \equiv \Sigma^*_s \cup \Sigma^*_{st} \cup \Sigma_t$ and $\Sigma' \equiv \Sigma^*_s \cup \Sigma^*_{st} \cup \Sigma'_t$. Since DE-equivalence implies CQ equivalence, in this section we always assume that two mappings whose DE-equivalence is under consideration have exactly the same set of s-t tgds and source egds.

As a consequence, by "differing dependency" we will always assume a target one. We also notice that the following simple proposition holds:

PROPOSITION 7.2. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma = \Sigma_s \cup \Sigma_{st} \cup \Sigma_t)$ be a mapping. For any dependency τ over $\mathbf{T}$, $\Sigma \models \tau$ iff $\Sigma_t \models \tau$.*

The following theorem can be illustrated by the mappings $\mathcal{M}$ and $\mathcal{M}_1$ from the Example 1.1.

THEOREM 7.3. *Let $\Sigma = \Sigma_s \cup \Sigma_{st} \cup \Sigma_t$ and $\Sigma' = \Sigma_s \cup \Sigma_{st} \cup \Sigma'_t$ be schema mappings with terminating chase property where the target dependencies consist of egds and tgds. If Σ and Σ' satisfy the AUC, then $\Sigma \equiv_{DE} \Sigma'$ holds.*

It turns out that for DE-equivalent mappings, the AUC can be reformulated in terms of unsatisfiability in either mapping:

THEOREM 7.4. *Let $\Sigma = \Sigma_s \cup \Sigma_{st} \cup \Sigma_t$ and $\Sigma' = \Sigma_s \cup \Sigma_{st} \cup \Sigma'_t$ be two DE-equivalent mappings. For any differing dependency $\tau \in \Sigma \cup \Sigma'$, let $\Sigma_\tau \in \{\Sigma, \Sigma'\}$ denote the mapping containing τ, and let $\Sigma_{\neg\tau}$ be the other mapping, $\Sigma_{\neg\tau} \not\models \tau$. Then, the following two statements are equivalent:*

1. *For any differing dependency $\tau \in \Sigma \cup \Sigma'$, the antecedent of τ is unsatisfiable in $\Sigma_{\neg\tau}$;*
2. *(AUC) For any differing dependency $\tau \in \Sigma \cup \Sigma'$, the antecedent of τ is unsatisfiable in Σ_τ.*

Theorem 7.4 will allow us to show, that for some useful class of mappings, the antecedent unsatisfiability condition is also necessary for DE-equivalence. Namely, this is the case for mappings in which target dependencies are *constant-free and connected*:

DEFINITION 7.4. *A conjunctive query is said to be connected if so is its Gaifman graph (that is, a graph in which nodes are query atoms and an edge is drawn between two nodes whenever the respective atoms share a variable).*

An egd is connected if its antecedent is. Finally, a tgd is connected if the conjunction of its antecedent and conclusion is: both the antecedent and the conclusion are connected and share at least one variable.

THEOREM 7.5. *Consider the mappings $\Sigma = \Sigma_s \cup \Sigma_{st} \cup \Sigma_t$ and $\Sigma' = \Sigma_s \cup \Sigma_{st} \cup \Sigma'_t$ in which both Σ_t and Σ'_t are constant-free and connected. Then, $\Sigma \equiv_{DE} \Sigma'$ implies that the two mappings satisfy the antecedent unsatisfiability condition.*

PROOF (SKETCH). Assume that $\Sigma \equiv_{DE} \Sigma'$ holds, but the antecedent unsatisfiability condition is violated: that is, there exists a differing dependency δ in either of the mappings, with the antecedent satisfiable in the mapping which contains δ. Then, by Theorem 7.4, also the following statement holds: W.l.o.g. there exists $\tau \in \Sigma_t$ such that $\Sigma' \not\models \tau$ and the antecedent $\varphi(\bar{x})$ of τ is satisfiable in Σ'. Now if τ and all dependencies in Σ'_t are constant-free and connected, we derive a contradiction by showing $\Sigma \not\equiv_{DE} \Sigma'$:

Indeed, assume there exists a source instance I such that $\varphi(\bar{x})$ is satisfied in $J = chase(I, \Sigma')$. Let $[\varphi]$ be a database isomorphic to $\varphi(\bar{x})$: each atom of $\varphi(\bar{x})$ is turned into a fact in $[\varphi]$ by instantiating the variables with distinct labelled nulls not occurring in J.

Based on $[\varphi]$, we now build a counter-example to $\Sigma \equiv_{DE} \Sigma'$. We first note that the chase with both Σ and Σ' succeeds on $[\varphi]$, as the target dependencies in both mappings are constant-free, and $dom([\varphi])$ consists solely of labeled nulls.

Consider the instance $J_\varphi = chase([\varphi], \Sigma')$. One can show the following three properties, justifying that the instance $J' = J \cup J_\varphi$ is exactly a counter-example to DE-equivalence of Σ and Σ':

a. $J' \not\models \Sigma_t$ and hence, $J' \notin Sol(I, \Sigma)$.
b. $J' \models \Sigma'_t$. Since $J \in Sol(I, \Sigma')$, we have $J' \in Sol(I, \Sigma')$.
c. There exists a homomorphism from J' onto any other solution for I under Σ'. Hence, $J' \in UnivSol(I, \Sigma')$. □

Theorem 7.5 allows one to reduce the data exchange equivalence of mappings with terminating chase property, containing constant-free and connected target dependencies, to testing the satisfiability of conjunctive queries according to Definition 7.1. This is good news, since the satisfiability of a conjunctive query boils down to evaluating the query against the chase of some fixed instance.

DEFINITION 7.5. *A* singleton-domain database $I^1_{\mathbf{S}}$ *for schema $\mathbf{S}$ contains a single atom $R(c, \ldots c)$ for each relation R in $\mathbf{S}$, where c is some constant: $dom(I^1_{\mathbf{S}}) = \{c\}$.*

THEOREM 7.6. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be schema mapping where Σ consists of source egds, source-to-target tgds, and target egds and tgds, possessing the terminating chase property. A Boolean conjunctive query q is satisfiable in Σ iff $chase(I^1_{\mathbf{S}}, \Sigma) \models q$.*

PROOF. There is a homomorphism h (not preserving constants) from any source database I onto $I^1_{\mathbf{S}}$. Thus, any dependency which fires in the chase of I is also applicable when $I^1_{\mathbf{S}}$ is chased, and h can be extended to $h'\colon chase(I, \Sigma) \to chase(I^1_{\mathbf{S}}, \Sigma)$ by extending the domain of h to the nulls introduced by the chase of I with Σ. Hence, for every I, $chase(I, \Sigma) \models q$ only if $chase(I^1_{\mathbf{S}}, \Sigma) \models q$. The "if" direction is trivial. □

Finally, we bring together the results of this section, of Section 6 and Lemma 2.1 to establish the DE-equivalence as a useful tool for optimizing practically relevant schema mappings. We say that

a schema mapping possesses the *polytime-terminating chase property*, if on any source instance, the chase with this mapping terminates in polynomial time. So are, e.g., the super-weakly-acyclic mappings [19], and inductively-restricted mappings [20].

THEOREM 7.7. *DE-equivalence of schema mappings with target dependencies consisting of FDs and IDs, and possessing the terminating chase property is decidable.*

Furthermore, suppose that these schema mappings possess the polytime-terminating chase property and that the number of variables in the antecedent resp. conclusion of s-t tgds in the mappings is bounded by some constant b. Then DE-equivalence of such mappings can be decided in polynomial time.

PROOF. First note that FDs and IDs are always connected and constant-free. Then, given the mappings $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\Sigma' = \Sigma'_{st} \cup \Sigma'_t$ where both Σ_t and Σ'_t are restricted to FDs and IDs and possess the terminating chase property, the following procedure decides $\Sigma \equiv_{DE} \Sigma'$:

1. Transform Σ and Σ' into the form $\Sigma_s \cup \hat{\Sigma}_{st} \cup \Sigma_t$ resp. $\Sigma'_s \cup \hat{\Sigma}'_{st} \cup \Sigma'_t$ using the PROPAGATE^E procedure. By Theorem 6.4, if $\Sigma_s \not\equiv \Sigma'_s$ or $\hat{\Sigma}_{st} \not\equiv \hat{\Sigma}'_{st}$, conclude $\Sigma \not\equiv_{DE} \Sigma'$.
2. Conclude $\Sigma \equiv_{DE} \Sigma'$ iff the pair (Σ, Σ') meets the antecedent unsatisfiability condition.

The polynomial-time upper bound is due to the fact that the mappings have the polytime-terminating chase property and the dependency size is bounded. For the target dependencies this bound is implicit in the size of the schema, which we consider as fixed. □

The previous result extends easily to mapping optimization with respect to redundancy elimination and subset-minimality. Namely, the following theorem can be easily shown:

THEOREM 7.8. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping with target FDs and IDs, possessing the polytime-terminating chase property. Moreover, assume that the number of antecedent resp. conclusion variables in the s-t tgds of Σ is limited by some constant b. Then, the following problems can be decided in polynomial time:*

- *checking if a dependency $\tau \in \Sigma$ is redundant w.r.t. DE-equivalence, and therefore,*
- *checking if Σ is subset-minimal.*

Thus, in terms of complexity, DE equivalence, while offering strictly higher optimization potential, is no worse than logical equivalence (studied in [15]) in the setting of Theorem 7.7. Note also, that the Theorems 7.7 and 7.8 hold for a class of mappings which is broader than FDs and IDs, namely constant-free and connected target tgds and egds. For such mappings, we may also investigate other useful minimization tasks which can be immediately seen to be decidable: for example, testing if any atom can be eliminated from a given dependency.

8. CONCLUSION

We have studied DE- and CQ-equivalence of schema mappings consisting of s-t tgds as well as target tgds and/or target egds. We have shown that the equivalence itself as well as various natural optimization tasks under these notions of equivalence are undecidable. That is, in these important aspects, the two notions of equivalence display the same behaviour. However, we have also identified a significant difference between them: CQ-equivalence remains undecidable even if the target dependencies are restricted to key dependencies while DE-equivalence becomes decidable.

As far as future work is concerned, we would like to search for further decidable fragments, in particular of DE-equivalence. Conversely, also the undecidability results (in particular for DE-equivalence) should be extended to further special cases. Another important direction of future work is to extend the investigation into the decidability/undecidability of DE- and CQ-equivalence and of related optimization tasks to other kinds of mappings like schema mappings consisting of Second-Order tgds.

9. REFERENCES

[1] M. Arenas, R. Fagin, and A. Nash. Composition with target constraints. In *Proc. ICDT'10*, pages 129–142. ACM, 2010.

[2] M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Inverting schema mappings: Bridging the gap between theory and practice. *PVLDB*, 2(1):1018–1029, 2009.

[3] M. Arenas, J. Pérez, and C. Riveros. The recovery of a schema mapping: Bringing exchanged data back. *ACM Trans. Database Syst.*, 34(4), 2009.

[4] C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.

[5] P. A. Bernstein. Applying model management to classical meta data problems. In *Proc. CIDR'03*, 2003.

[6] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *Proc. PODS'08*, pages 149–158. ACM, 2008.

[7] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive query plans for data integration. *J. Log. Prog.*, 43(1), 2000.

[8] R. Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.

[9] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[10] R. Fagin, P. G. Kolaitis, A. Nash, and L. Popa. Towards a theory of schema-mapping optimization. *PODS'08*, ACM.

[11] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM Trans. Dat. Syst.*, 30(1):174–210, 2005.

[12] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.

[13] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Reverse data exchange: coping with nulls. *PODS'09*, pages 23–32, ACM.

[14] G. Gottlob and C. H. Papadimitriou. On the complexity of single-rule datalog queries. *Inf. Comput.*, 183(1), 2003.

[15] G. Gottlob, R. Pichler, and V. Savenkov. Normalization and optimization of schema mappings. *PVLDB*, 2(1), 2009.

[16] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data integration: The teenage years. In *Proc. VLDB'06*.

[17] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

[18] J. Madhavan and A. Y. Halevy. Composing mappings among data sources. In *Proc. VLDB'03*, pages 572–583, 2003.

[19] B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.

[20] M. Meier, M. Schmidt, and G. Lausen. On chase termination beyond stratification. *PVLDB*, 2(1):970–981, 2009.

[21] S. Melnik. *Generic Model Management: Concepts and Algorithms*, volume 2967 of *LNCS*. Springer, 2004.

[22] A. Nash, P. A. Bernstein, and S. Melnik. Composition of mappings given by embedded dependencies. *ACM TODS*, 32(1):4, 2007.

[23] O. Shmueli. Equivalence of datalog queries is undecidable. *J. Log. Prog.*, 15(3):231–241, 1993.