On the Undecidability of the Equivalence of Second-Order Tuple Generating Dependencies *

Ingo Feinerer, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov

Vienna University of Technology
{feinerer|pichler|sallinger|savenkov}@dbai.tuwien.ac.at

Abstract. Second-Order tuple generating dependencies (SO tgds) were introduced by Fagin et al. to capture the composition of simple schema mappings. Testing the equivalence of SO tgds would be important for applications like model management and mapping optimization. However, we prove the undecidability of the logical equivalence of SO tgds. Moreover, under weak additional assumptions, we also show the undecidability of a relaxed notion of equivalence between two SO tgds, namely the so-called conjunctive query equivalence.

1 Introduction

Schema mappings play an important role in several areas of database research, notably in data integration [13], data exchange [9], peer data management [14], and model management [6]. A schema mapping is given by two schemas, called the *source schema* and the *target schema*, as well as a set of *dependencies* describing the relationship between the source and target schema. The most fundamental form of schema mappings are mappings defined by a set of source-to-target tuple generating dependencies (s-t tgds): They are First-Order formulae of the form $\forall \boldsymbol{x}(\varphi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y}\psi(\boldsymbol{x}, \boldsymbol{y}))$, where the antecedent $\varphi(\boldsymbol{x})$ is a conjunctive query (CQ) over the source schema and the conclusion $\psi(\boldsymbol{x}, \boldsymbol{y})$ is a CQ over the target schema. Intuitively, such an s-t tgd defines a constraint that the presence of certain tuples in the source database I (namely those in the image of some homomorphism h from $\varphi(\boldsymbol{x})$ to I) enforce the presence of certain tuples in the target database J (s.t. h can be extended to a homomorphism from $\psi(\boldsymbol{x}, \boldsymbol{y})$ to J).

Several algebraic operators [6, 15] on schema mappings have been intensively studied in recent time like computing inverses [8, 3, 2] and composing schema mappings [7, 12, 14, 16]. Our work is rather related to the composition operator. Fagin et al. proved that s-t tgds are not powerful enough to express the composition of two mappings defined by s-t tgds [12]. To remedy this defect, so-called Second-Order tuple generating dependencies (SO tgds) were introduced in [12]. SO tgds extend s-t tgds by existentially quantified function-variables and equalities of (possibly functional) terms. Details and formal definitions are given in Section 2. It was shown in [12] that SO tgds capture exactly the closure under composition of mappings defined by s-t tgds.

Example 1 ([12]). Consider the following three *schemas*. Let S_1 consist of the unary relation symbol $\mathsf{Emp}(\cdot)$ of employees. Schema S_2 consists of a single binary relation

* This work was supported by the Vienna Science and Technology Fund (WWTF), project ICT08-032. Savenkov was supported by the Erasmus Mundus ECW Program of the EU.

symbol $\mathsf{Mgr}'(\cdot,\cdot)$ that associates each employee with a manager. Schema S_3 consists of a similar binary relation symbol $\mathsf{Mgr}(\cdot,\cdot)$ that is intended to provide a copy of Mgr' and an additional unary relation symbol $\mathsf{SelfMgr}(\cdot)$ to store employees who are their own manager. Consider the *mappings* $\mathcal{M}_{12} = (S_1, S_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (S_2, S_3, \Sigma_{23})$ with $\Sigma_{12} = \{\forall e(\mathsf{Emp}(e) \rightarrow \exists m\, \mathsf{Mgr}'(e,m))\}$ and $\Sigma_{23} = \{\forall e, m(\mathsf{Mgr}'(e,m) \rightarrow \mathsf{Mgr}(e,m)), \forall e(\mathsf{Mgr}'(e,e) \rightarrow \mathsf{SelfMgr}(e))\}$. We are looking for the *composition* of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. It can be verified that this composition can be expressed by the SO tgd $\sigma = \exists f\,(\forall e\,(\mathsf{Emp}(e) \rightarrow \mathsf{Mgr}(e, f(e))) \wedge \forall e\,(\mathsf{Emp}(e) \wedge (e = f(e)) \rightarrow \mathsf{SelfMgr}(e)))$.

In this paper, we want to study the *equivalence of SO tgds*. Note that the question of equivalence naturally arises in several scenarios. Figure 1 illustrates a *model evolution* scenario, where data structured under some schema S is first migrated to a database with schema T and then further transformed to meet schema U. Now suppose that there exists an alternative migration path from schema S via T' to schema U. The question if the two migration paths yield the same result comes down to checking if the dependencies σ and σ' (which represent the respective mapping compositions) are equivalent. Actually, Figure 1 can also be thought of as illustrating a *peer data management* scenario, where some peer with data structured according to S provides part of its data to some other peer with schema T (resp. T'). The latter peer in turn passes this data on to yet another peer with schema U. Now suppose that a user may access the data only at the peer with schema U. What happens if some link in this peer data network is broken, say the one corresponding to mapping Σ_{23}? Will the path of mappings from S via T' to U still give the user full access to the data provided by the peer with schema S? Testing the equivalence of σ and σ' is thus crucial for answering questions of redundancy and reliability in a peer data network.

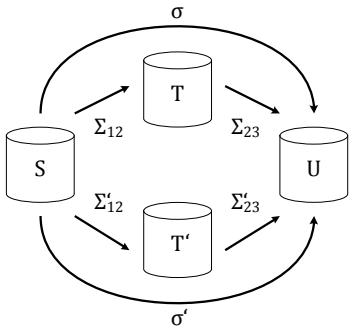

Fig. 1. Mapping compositions.

The equivalence of mappings is also fundamental to mapping optimization. As mentioned in [10], optimizing a mapping ultimately means replacing the mapping by an "*equivalent*" one with better (computational) properties. This raises the question of how the "*equivalence*" of two mappings should be defined. Since dependencies are logical formulae, the most natural notion of equivalence is *logical equivalence*. In this paper, we show that logical equivalence of SO tgds is undecidable. In order to allow for more flexibility in optimizing mappings, Fagin et al. introduced relaxed notions of equivalence [10]. In particular, the potential of *conjunctive query* (CQ) equivalence for optimizing several kinds of mappings was studied in [10]. Intuitively, two mappings are CQ-equivalent, if conjunctive queries posed against the target database yield the same result for both mappings (for details, see Section 2). For instance, it was shown in [12] that the composition of the mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ in Example 1 cannot be represented by an SO tgd without equalities in the antecedent. On the other hand, σ in Example 1 is CQ-equivalent to $\sigma' = \exists f\,(\forall e\,(\mathsf{Emp}(e) \rightarrow \mathsf{Mgr}(e, f(e)))$. Under the

weak additional assumption that the source schema may have key dependencies, we shall prove the undecidability of CQ-equivalence of SO tgds.

Organization of the paper and summary of results. In Section 2, we recall some basic notions and results. A conclusion and an outlook to future work are given in Section 5. The main results of the paper are detailed in Sections 3 and 4, namely:

• *Logical equivalence.* In Section 3 we prove that the logical equivalence of SO tgds is undecidable. This result is easily obtained via a previous undecidability result in the context of inverse schema mappings [3].

• *Logical equivalence vs. CQ-equivalence.* In Section 3, we also present a different proof strategy for the undecidability of the logical equivalence of two SO tgds in order to identify an important difference between logical equivalence and CQ-equivalence. More precisely, we show that logical equivalence of two SO tgds remains undecidable even if the two SO tgds are already known to be CQ-equivalent.

• *CQ-equivalence.* In Section 4 we prove the undecidability of CQ-equivalence of SO tgds if the source schema may have key dependencies. The proof is by reduction from the domino problem. As a by-product of this proof, we also get the undecidability of logical equivalence of SO tgds without equalities.

Due to lack of space, proofs are sketched. Details will be provided in the full version.

2 Preliminaries

Schemas and instances. 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 R^I to denote the relation of R in I. We only consider finite instances here.

Schema mappings. Let $\mathbf{S} = \{S_1, \ldots, S_n\}$ and $\mathbf{T} = \{T_1, \ldots, T_m\}$ be schemas with no relation symbols in common. 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 logical formulae called dependencies expressing the relationship between $\mathbf{S}$ and $\mathbf{T}$.

Instances over $\mathbf{S}$ (resp. $\mathbf{T}$) are called *source* (resp. *target*) *instances*. We write $\langle \mathbf{S}, \mathbf{T} \rangle$ to denote the schema $\{S_1, \ldots, S_n, T_1, \ldots, T_m\}$. If I is a source instance and J a target instance, then $\langle I, J \rangle$ is an instance of the schema $\langle \mathbf{S}, \mathbf{T} \rangle$.

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

Dependencies. Source-to-target tuple generating dependencies (as already defined in the introduction) are logical formulae of the form $\forall \boldsymbol{x}(\varphi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y}))$. We write $\boldsymbol{x}$ for a tuple $(x_1, \ldots, x_n)$. However, by slight abuse of notation, we also refer to the set $\{x_1, \ldots, x_n\}$ as $\boldsymbol{x}$. Hence, we may use expressions like $x_i \in \boldsymbol{x}$ or $\boldsymbol{x} \subseteq X$, etc.

A *key dependency* over schema $\mathbf{R}$ is of the form $\forall \boldsymbol{x}(R(\boldsymbol{u}, y, \boldsymbol{v}) \wedge R(\boldsymbol{u}, z, \boldsymbol{w}) \rightarrow y = z)$ where R denotes a relation symbol in $\mathbf{R}$ with $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \subseteq \boldsymbol{x}$ and $y, z \in \boldsymbol{x}$.

A *second-order tuple generating dependency (SO tgd)* is a logical formula of the form $\exists \boldsymbol{f}((\forall \boldsymbol{x}_1(\varphi_1 \rightarrow \psi_1)) \wedge \cdots \wedge (\forall \boldsymbol{x}_n(\varphi_n \rightarrow \psi_n)))$ where (1) each member of $\boldsymbol{f}$ is a function symbol, (2) each φ_i is a conjunction of atomic formulas of the form

$S(y_1, \ldots, y_k)$ (with $S \in \mathbf{S}$ and $y_j \in \boldsymbol{x}_i$), and equalities of the form $t = t'$ (with t and t' terms based on $\boldsymbol{x}_i$ and $\boldsymbol{f}$), (3) each ψ_i is a conjunction of atomic formulas of the form $T(t_1, \ldots, t_\ell)$ (with $T \in \mathbf{T}$ where $t_1, \ldots, t_\ell$ are terms based on $\boldsymbol{x}_i$ and $\boldsymbol{f}$), and (4) each variable in $\boldsymbol{x}_i$ appears in some atomic formula of φ_i.

When dealing with instances $\langle I, J \rangle$ in the context of SO tgds the domain of source instances I is allowed to consist only of constants (i.e., grounded) whereas target instances J may contain functional terms which can be treated like fresh variables, also called labelled nulls. [12]

Homomorphisms. Let I, I' be instances. A *homomorphism* $h\colon I \to I'$ is a mapping s.t. (1) whenever $R(\boldsymbol{x}) \in I$, then $R(h(\boldsymbol{x})) \in I'$, 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*.

If a homomorphism $h\colon I \to I'$ is invertible, s.t. h^{-1} is also 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* $core(J) \cong core(J')$.

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, their core is unique up to isomorphism [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.

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

Chase. A target instance for a given source instance and a schema mapping can be computed by the *chase* [4]. For SO tgds, the original chase procedure of [4] has to be modified since homomorphisms now have to take the equalities in the antecedents into account. Formally, a mapping h from a conjunct $C_i = \forall \boldsymbol{x}(\varphi_i \to \psi_i)$ of an SO tgd to an instance I is a homomorphism if for every relational atom $S(y_1, \ldots, y_k)$ in φ_i the tuple $(h(y_1), \ldots, h(y_k))$ is in S^I and for every equality $t = t'$ we have $h(t) = h(t')$.

Let $\langle I, J \rangle$ be an instance of a schema $\langle \mathbf{S}, \mathbf{T} \rangle$ and let $C_i = \forall \boldsymbol{x}(\varphi_i \to \psi_i)$ be an SO conjunct. If there is a homomorphism h of φ_i into the instance I, then the conjunct C_i can be applied to $\langle I, J_1 \rangle$ with homomorphism h. I.e., for every target atom $T(t_1, \ldots, t_\ell)$ of ψ_i the tuple $(h(t_1), \ldots, h(t_\ell))$ is added to the T relation and the union with J_1 is taken yielding a new instance J_2. The instance $\langle I, J_2 \rangle$ is now called the result of applying C_i to $\langle I, J_1 \rangle$ with h. The actual application is called a *chase step*.

A *chase sequence* is a finite sequence of chase steps. The *chase* of $\langle I, \emptyset \rangle$ with an SO tgd Σ is a chase sequence until no conjuncts can be applied anymore. The result of this chase is denoted by $chase(I, \Sigma)$. Chasing $\langle I, \emptyset \rangle$ with SO tgds can be done in polynomial time w.r.t. the size of the source instance and results in a universal solution [12].

Equivalence of schema mappings. 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. This is the case iff $Sol(I, \mathcal{M}) = Sol(I, \mathcal{M}')$ holds for every source instance I.

$\mathcal{M}$ and $\mathcal{M}'$ are *conjunctive-query 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 $core(I, \mathcal{M}) = core(I, \mathcal{M}')$. Note that 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 $\mathcal{M}$ and $\mathcal{M}'$. The above characterization via the core, which is more convenient for our purposes here, was proved to be equivalent in [10].

3 Undecidability of Logical Equivalence

The undecidability of logical equivalence of two SO tgds follows easily from a previous undecidability result in the context of inverse schema mappings [3].

Theorem 1. *Let τ and τ' be two SO tgds. It is undecidable whether $\tau \equiv \tau'$, i.e. that τ is logically equivalent to τ'.*

Proof. (Sketch).[1] In [8], a schema mapping $\mathcal{M}_2$ is defined to be an inverse of another mapping $\mathcal{M}_1$ if their composition $\mathcal{M}_1 \circ \mathcal{M}_2$ is logically equivalent to the identity mapping, i.e., a set of s-t tgds of the form $(\forall \boldsymbol{x})\big(P(\boldsymbol{x}) \rightarrow \hat{P}(\boldsymbol{x})\big)$ for every source relation P. In [3, Corollary 9.5] the authors show that the following problem is undecidable: Given mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ specified by s-t tgds, check whether $\mathcal{M}_2$ is an inverse of $\mathcal{M}_1$.

In other words, checking if the composition $\mathcal{M}_1 \circ \mathcal{M}_2$ is equivalent to the identity mapping is undecidable. Clearly, a set of s-t tgds can be represented as an SO tgd. Let τ denote the SO tgd corresponding to the set of s-t tgds of the identity mapping. Likewise, $\mathcal{M}_1 \circ \mathcal{M}_2$ can be represented as an SO tgd, say τ'. Then $\tau \equiv \tau'$ holds iff $\mathcal{M}_2$ is an inverse of $\mathcal{M}_1$. Hence, the logical equivalence of two SO tgds is undecidable. □

We now want to strengthen the above result by relating logical equivalence to CQ-equivalence. Below, we show that logical equivalence of two SO tgds remains undecidable even if the two SO tgds are already known to be CQ-equivalent. To this end, we adopt a different proof strategy which goes by reduction from the following variant of the Halting problem: Given a Turing machine *TM*, does *TM* halt when run on empty input? Given an arbitrary instance *TM* of the Halting problem, we have to construct two SO tgds τ and τ', s.t. the following equivalence holds: *TM* halts on empty input $\Leftrightarrow$ $\tau \not\equiv \tau'$. A major challenge of this reduction is that SO tgds do not directly provide us with recursion or with a means to enforce equalities.

[1] We thank the anonymous referee of AMW 2011 who pointed out this proof.

Schemas. Let *TM* be a Turing machine (Q, A, δ, q_0) with set of states $Q = \{0, \ldots, m\}$, tape alphabet $A = \{0, \ldots, n\}$, transition function $\delta : Q \times A \to Q \times A \times \{\leftarrow, \rightarrow\}$, and initial state q_0. W.l.o.g., assume that $q_0 = 0$ and let 1 denote the halting state. Moreover, let 1 denote the tape starting symbol $\triangleright$ and let 0 denote the blank symbol $\sqcup$.

We represent the state, tape and cursor of a Turing machine configuration at time t and tape location l by the **function symbols** $\mathsf{state}_q(t)$, $\mathsf{tape}_a(t, l)$, and $\mathsf{cursor}(t, l)$ (where $q \in Q$, $a \in A$). Each of these functions is intended to have Boolean range, e.g. $\mathsf{tape}_a(t, l) = 1$ indicates that at time t, the symbol a is stored on the tape at location l, and $\mathsf{tape}_a(t, l) = 0$ otherwise. We will ensure this Boolean range using our SO tgds.

As **source schema**, we use $\mathsf{root}(\cdot)$ and $\mathsf{chain}(\cdot, \cdot)$, which is intended to describe a linear order starting at $\mathsf{root}(r)$ and continued by $\mathsf{chain}(\cdot, \cdot)$ atoms, i.e. source instance I should have the form $I = \{\mathsf{root}(0), \mathsf{chain}(0, 1), \mathsf{chain}(1, 2), \ldots\}$. Still, we must make sure through our SO tgds that what is described by root and chain does not deviate from a linear order in a way that adversely affects our simulation.

We define our **target schema** to consist of the relation symbol $\mathsf{range}(\cdot)$, which is intended to describe the range of the functions we defined above. Furthermore, we use the target relation symbol halt to indicate a halting computation as well as to signify an error condition. That is, an intended target instance should have the form $J = \{\mathsf{range}(0), \mathsf{range}(1)\}$ and possibly include $\mathsf{halt}()$.

Dependencies. Given an arbitrary Turing machine *TM*, we define the SO tgds τ and τ' as a "big" conjunction consisting of several groups of conjuncts (i.e., implications). As we will see, it is possible to construct an SO tgd that simulates a computation of *TM*.

We start by describing **initialization conjuncts** which are used to encode the initial configuration of the *TM* on the empty input tape. Since we cannot enforce equalities on the right-hand side of implications, all the implications are formulated in such a way that taking the *wrong* value will lead to an error condition indicated by $\mathsf{halt}()$. E.g., we want to enforce the initial state $\mathsf{state}_0(r) = 1$. Therefore, in the case that $\mathsf{state}_0(r) = 0$ holds, we enforce $\mathsf{halt}()$ in the target instance to indicate this error condition:

- $\mathsf{root}(r) \wedge \mathsf{state}_0(r) = 0 \to \mathsf{halt}()$

Note that this does not yet cover the case that $\mathsf{state}_0(r) \notin \{0, 1\}$. We construct other implications which define the rest of the initial configuration analogously.

We now define **transition conjuncts**, which implement the transition function of *TM*. In what follows, we write $dom(z)$ as a short-hand for a predicate that is satisfied by instantiating z to an arbitrary constant occurring in the source instance, i.e., $dom(z)$ has to be replaced by $\mathsf{root}(z)$, $\mathsf{chain}(z, \_)$, or $\mathsf{chain}(\_, z)$. Such a predicate is needed to fulfil the safety condition of SO tgds. We first define a common part of the following implications, matching the current state q and the current tape symbol a.

$$\varphi := \mathsf{state}_q(t) = 1 \wedge \mathsf{cursor}(t, l) = 1 \wedge \mathsf{tape}_a(t, p) = 1 \wedge \mathsf{chain}(t, t') \wedge dom(l)$$

Suppose that the transition function δ contains the transition $\delta(q, a) = (q', a', \leftarrow)$. Then the SO tgds τ and τ' contain the following conjuncts:

- $\varphi \wedge \mathsf{state}_{q'}(t') = 0 \to \mathsf{halt}()$
- $\varphi \wedge \mathsf{tape}_{a'}(t', l) = 0 \to \mathsf{halt}()$
- $\varphi \wedge \mathsf{chain}(l', l) \wedge \mathsf{cursor}(t', l') = 0 \to \mathsf{halt}()$

While the transition conjuncts define what changes, we now construct **inertia conjuncts** which describe the function values of what remains the same. In particular, we can construct (but for space reasons omit here) conjuncts, which define that the cursor is *not* located at positions that are (a) more than one move away from the previous cursor location, (b) at the same location, or (c) at the location opposite to the cursor movement.

Furthermore, we need to state that the tape symbol remains the same wherever the cursor is not located. That is, for each $q \in Q$, $a \in A$:

- $\mathsf{state}_q(t) = 1 \wedge \mathsf{cursor}(t, l) = 0 \wedge \mathsf{tape}_a(t, l) = 1 \wedge dom(l) \wedge$
  $\mathsf{chain}(t, t') \wedge \mathsf{tape}_a(t', l) = 0 \rightarrow \mathsf{halt}()$

Finally, we also have to construct **uniqueness conjuncts**, ensuring that at each time t, there is exactly one $\mathsf{state}_q(t)$ and at each time t and location l exactly one $\mathsf{tape}_a(t, l)$ whose function value is 1.

Guarding conjuncts. It is clear that, since the domain is in general non-Boolean, we cannot prevent functions state_q, tape_a and cursor to take a range outside of $\{0, 1\}$ for arbitrary interpretations. However, the structure of our reduction will show that we have to either, for a given interpretation of the function symbols, define the interpretation of the *relation* symbols, or for a given interpretation of the relation symbols, define the interpretation of the *function* symbols.

So if we define the functions, we can guarantee Boolean domain ourselves. If we define the relations, and in particular the target relations, we can use **function guarding conjuncts** to materialize the range in the target:

- $dom(x) \wedge dom(y) \wedge \mathsf{state\_}(x) = y \rightarrow \mathsf{range}(y)$

Analogous conjuncts are introduced for tape and cursor. This ensures the Boolean range of our functions, given appropriately constructed target relations ($\mathsf{range}^J = \{0, 1\}$).

Now that we have sufficient control over the functions, we still need to gain some control over the source instance, since our simulation is intended to work on a linear order described by root and chain. We therefore define **source guarding conjuncts**. What we can directly enforce is that the chain relation includes no self-loops and never returns to any root using the following conjuncts:

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

However, what we cannot directly exclude is that some element x of the chain has two distinct predecessors $px \neq px'$ with $\mathsf{chain}(px, x)$ and $\mathsf{chain}(px', x)$. In particular, we do not have equality on the right-hand side, which would allow us to require $px = px'$. Nor can we use a derivation of $\mathsf{halt}()$ when there are two predecessors $\mathsf{chain}(px, x)$ and $\mathsf{chain}(px', x)$ because we cannot guarantee that $px \neq px'$ on the left-hand side.

But what we can do is control the *effect* of such multiple predecessors (or successors). Notably, if these multiple predecessors or successors actually have no effect on our simulation, then we have no problem with these discrepancies from linear order, since the simulation is doing the correct thing anyway. Altogether, we cannot detect *all* deviations from a linear order, e.g., two distinct $\mathsf{root}(r)$ atoms cannot be ruled out. But we can gain just enough control to ensure the correctness of our reduction.

Halting detection. We now construct the two SO tgds for our reduction, namely such that *TM* halts $\Leftrightarrow \tau \not\equiv \tau'$. The first one, τ, simply consists of all the conjuncts we

constructed up to this point, that is simulation conjuncts and guarding conjuncts. The function symbols $\mathsf{tape}\_$, $\mathsf{state}\_$ and cursor are in the scope of an existential quantifier over the entire conjunction. We now define the **halting detection conjunct**:

- $dom(t) \wedge \mathsf{state}_1(t) = 1 \rightarrow \mathsf{halt}()$

and construct τ' to be the same as τ plus this halting detection conjunct. So overall, both τ and τ' simulate the computation of Turing machine *TM*, but only τ' indicates a halting state by enforcing $\mathsf{halt}()$.

Theorem 2. *Let τ and τ' be two SO tgds. It is undecidable whether $\tau \equiv \tau'$ (i.e. τ is logically equivalent to τ'), even when $\tau \equiv_{CQ} \tau'$ (i.e. τ is CQ-equivalent to τ').*

Proof. (Sketch). We constructed τ' as τ with an additional conjunct, therefore it is sufficient to show that *TM* halts $\Leftrightarrow \tau \not\models \tau'$. We show both directions separately. The challenging part of the reduction is that, in one direction we have control over the functions, but not over the relations, and the converse for the other direction.

Assume that TM halts. Then we have to find an instance $K = (I, J)$ such that $K \models \tau$ but $K \not\models \tau'$. Since *TM* halts, there is a computation of length n that ends in the halting state. We construct the instance $K = (I, J)$ as

$$I = \{\mathsf{root}(1), \mathsf{chain}(1,2), \ldots, \mathsf{chain}(n-1, n)\} \quad J = \{\mathsf{range}(0), \mathsf{range}(1)\}$$

Then $K \models \tau$ is clearly the case. What needs to be shown is that $K \not\models \tau'$. In particular, $K \not\models \tau'$ means that for *arbitrary* $\boldsymbol{f}$, we have $(K, \boldsymbol{f}) \not\models \varphi'$. That is, we constructed K, but must deal with arbitrary $\boldsymbol{f}$ for φ'. We proceed by case distinction, assuming $v_{t,l,a}$ is the correct value of $\mathsf{tape}_a(t, l)$ according to the transition function of *TM*:

1. $\mathsf{tape}_a(t, l) = v_{t,l,a}$ for all t, l, a, that is, it correctly represents the *TM* computation
2. $\mathsf{tape}_a(t, l) = \neg v_{t,l,a}$ for some t, l, a, that is, the simulation violates *TM*
3. For some $\mathsf{tape}_a(t, p) = y$ we have that $y \notin \{0, 1\}$, i.e., non-Boolean range

and in each case show $K, \boldsymbol{f} \not\models \tau'$. This also holds for state and cursor.

Assume that TM does not halt. Given an arbitrary instance $K = (I, J)$, we have to show that $K \models \tau$ implies $K \models \tau'$. That is, we deal with arbitrary K, but only have to find a single $\boldsymbol{f}$ for φ'. Assume that *TM* does not halt. Then let $K = (I, J)$ be an arbitrary instance. We proceed by case distinction based on K:

1. J already contains an atom $\mathsf{halt}()$
2. Source instance I deviates from a linear order significantly for our simulation
3. J contains some $\mathsf{range}(c)$ atom with $c \notin \{0, 1\}$, i.e., allows non-Boolean range
4. All $\mathsf{range}(c)$ atoms in J have $c \in \{0, 1\}$

and in each case show $K, \boldsymbol{f} \models \tau'$.

Finally, note that the SO tgds τ and τ' in the above problem reduction are CQ-equivalent. The claim of the theorem thus follows. □

Equalities are fundamental in SO tgds in order to define the composition of two schema mappings [12, Theorem 5.4]. Equalities also play a central role in our undecidability proof of logical equivalence for SO tgds in Theorem 2 above. However, if we consider CQ-equivalence rather than logical equivalence, then the equalities are not strictly necessary anymore and can be removed: In [2] it is proved that every SO tgd is CQ-equivalent to a plain SO tgd (a class of SO tgds which do not contain equalities). As a consequence the previous proof idea cannot be reused and we need a set of new proof techniques to show undecidability also for CQ-equivalence.

4 Undecidability of Conjunctive Query Equivalence

We now show the undecidability of CQ-equivalence of SO tgds by a reduction from the Domino Problem (DP), which was first shown undecidable by Berger [5]. We use the following, slightly modified version of DP [1, p.414].

Definition 1. *A* domino system *(DS) is given by a set $\mathcal{D}$ of* domino types *defined as follows. Let $\mathcal{C}$ be a set of colours. Then, each domino type is a quadruple $\langle l, t, r, b\rangle$ of values from $\mathcal{C}$, corresponding respectively to the left, top, right and bottom colour of a square domino piece. Let $d.side$ denote an accessor function $\mathcal{D} \to \mathcal{C}$ returning the colour of d, corresponding to $side$. Then, the Domino Problem (DP) given by $\mathcal{D}$ asks if for each $m, n \in \mathbb{N}$ a* consistent tiling *of an $m \times n$ grid with the domino types from $\mathcal{D}$ exists. That is, if there is a function $t_{m,n}\colon \{1 \dots m\} \times \{1 \dots n\} \to \mathcal{D}$, such that the equality $t_{m,n}(i,j).right = t_{m,n}(i+1,j).left$ holds for $1 \leq i < m$ and $1 \leq j \leq n$ and $t_{m,n}(k,l).bottom = t_{m,n}(k,l+1).top$ for $1 \leq k \leq m$ and $1 \leq l < n$.*

Chains, proper instances, and coordinates. We model a grid using the source schema $\mathbf{S}$ consisting of two binary relations C_h and C_v intended to define a horizontal and vertical successor relation: For an instance I of $\mathbf{S}$, C_h^I and C_v^I denote the relations of I. Their intended form is $\{(a_0, a_1), (a_1, a_2) \dots (a_{n-1}, a_n)\}$, with $a_i \neq a_j$, for every $0 \leq i < j \leq n$. We define $a < b$ w.r.t. an order C, if $(a, b) \in C$ or $\exists x, C(x, b) \wedge a < x$. Taking the tuples of C as edges of a directed graph (which we call the *dual graph*), the order defined above corresponds to an *acyclic chain*. If each relation of an instance I over $\mathbf{S}$ contains a single acyclic chain, I is called *proper*. The size (m, n) of such a proper instance I is defined as $(|C_h|, |C_v|)$.

Domino tilings are modelled by the target relation $D/4$. Each of its four arguments is populated by a functional term $c^s(x, y)$ where c corresponds to the colour, s to the side of the domino piece, and the pair (x, y) defines the position in the grid. To save notation, we will omit parentheses when writing the terms in the relation D (e.g. the above term will be written as $c^s xy$). Given a fact $d \in D$, we define the *accessor function* $d.n$, returning the term in the n^{th} place of d, $1 \leq n \leq 4$.

Definition 2. *Let $\mathcal{D}$ be a DS with types $d_1 \dots d_n$. A simulation mapping $\Sigma_\mathcal{D}$ for $\mathcal{D}$ over the source schema $\mathbf{S} = \{C_h(\cdot,\cdot), C_v(\cdot,\cdot)\}$ and the target schema $\mathbf{T} = \{D(\cdot,\cdot,\cdot,\cdot)\}$ consists of an SO tgd $\tau_\mathcal{D}$ and four source key dependencies:*

$\epsilon_1\colon\ C_h(x, x_1) \wedge C_h(x, x_2) \to x_1 = x_2$ $\qquad$ $\epsilon_2\colon\ C_h(x_1, x) \wedge C_h(x_2, x) \to x_1 = x_2$
$\epsilon_3\colon\ C_v(x, x_1) \wedge C_v(x, x_2) \to x_1 = x_2$ $\qquad$ $\epsilon_4\colon\ C_v(x_1, x) \wedge C_v(x_2, x) \to x_1 = x_2$

The SO tgd $\tau_\mathcal{D}$ has the form $\exists \mathbf{c} \exists p^h \exists p^v \exists q^h \exists q^v (\tau_{d_1} \wedge \dots \wedge \tau_{d_n} \wedge \gamma_1 \wedge \gamma_2)$, where $\tau_{d_1} \dots \tau_{d_n}$ encode the corresponding domino types:

$\tau_{d_i}\colon\ C_h(x_1, x_2) \wedge C_v(y_1, y_2) \to D(l^h\, x_1\, y_2, t^v\, x_2\, y_1, r^h\, x_2\, y_2, b^v\, x_2\, y_2)$,

l, t, r, b denoting respectively the left, top, right and bottom colours of d_i. Superscripts distinguish the horizontal and vertical dimensions, so that there are at most two distinct functions in $\mathbf{c}$ for each colour in $\mathcal{D}$. Moreover, $\mathbf{c}$ contains neither $p^{h|v}$ nor $q^{h|v}$.

The conjuncts γ_1 and γ_2 are called guards *and have a form similar to τ_{d_i}:*

$\gamma_h\colon\ C_h(x_0, x_1) \wedge C_h(x_1, x_2) \wedge C_v(y_1, y_2) \to D(p^h\, x_1\, y_2, p^v\, x_2\, y_1, p^h\, x_2\, y_2, p^v\, x_2\, y_2)$
$\gamma_v\colon\ C_h(x_1, x_2) \wedge C_v(y_0, y_1) \wedge C_v(y_1, y_2) \to D(q^h\, x_1\, y_2, q^v\, x_2\, y_1, q^h\, x_2\, y_2, q^v\, x_2\, y_2)$

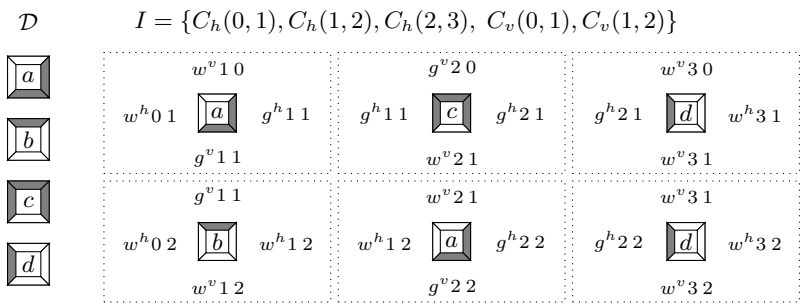

Fig. 2. Dotted rectangles are facts in $core(I, \Sigma_{\mathcal{D}})$, corresponding to a consistent tiling

Informally, given a proper source instance of size (m, n), the conjuncts $\tau_{d_1} \dots \tau_{d_{|\mathcal{D}|}}$ create a stack of $|\mathcal{D}|$ domino tiles at each position in the $m \times n$ grid. Additionally, the guards γ_h and γ_v create two more tiles with unique colours p and q on each position (i, j) with $i, j > 1$, or a single tile if $i = 1$ or $j = 1$. The guards tackle source instances with cycles instead of the linear order, as will be explained later.

For a source instance I, a D-fact d enforced by the SO tgd in a simulation mapping (that is, $d \in chase(I, \Sigma_{\mathcal{D}})$) has a form $D(l^h x_p\, y, t^v xy_p, r^h xy, b^v xy)$, such that $(x_p, x) \in C_h^I$, $(y_p, y) \in C_v^I$.

Definition 3. *Let $\mathcal{D}$ be a DS and I a proper instance I of size (m, n). A fact $d = D(l^h x_p y, t^v xy_p, r^h xy, b^v xy) \in chase(I, \Sigma_{\mathcal{D}})$ is associated with the domino type $\langle l, t, r, b\rangle$, and with the pair (x, y) of I-values, occurring as arguments in its two last terms. Moreover, let i be the ordinal of x w.r.t. the order defined by C_h^I and j be the ordinal of y w.r.t. C_v^I. We call the pair (i, j) the* coordinates *of d defining its* grid position.

Property 1. Let I be a proper source instance, and $\Sigma_{\mathcal{D}}$ the simulation mapping of a DS $\mathcal{D}$. Moreover, let d_1 and d_2 be two facts in $chase(I, \tau_{\mathcal{D}})$, d_1 associated with the pair of I-values (x_1, y_1), and d_2 with (x_2, y_2). Then, the following properties hold true:

1. $d_1.3 = d_2.1 \Rightarrow (x_1, x_2) \in C_h^I \wedge y_1 = y_2 \wedge \forall i \neq 3\ \forall j \neq 1\ d_1.i \neq d_2.j$.
2. $d_1.2 = d_2.4 \Rightarrow (y_1, y_2) \in C_v^I \wedge x_1 = x_2 \wedge \forall i \neq 4\ \forall j \neq 2\ d_1.i \neq d_2.j$.
3. $\exists i \in \{1...4\}\ d_1.i = d_2.i \Rightarrow x_1 = x_2 \wedge y_1 = y_2$.
4. $d_1.1 \neq d_2.2 \wedge d_1.3 \neq d_2.4 \wedge d_1.1 \neq d_2.4$.

Claim 1 stipulates, that whenever a term is shared between the 3^{rd} place of d_1 and the 1^{st} place of d_2, the facts correspond to horizontally adjacent grid positions (i.e., have first coordinates i and $i + 1$ and the same second coordinate) and can have no further terms in common. Claim 2 is an analogue of Claim 1 for the equality $d_1.4 = d_2.2$ and vertical adjacency: d_1 and d_2 must have coordinates (i, j) and $(i, j + 1)$. Claim 3 states that terms at respective positions can be only shared by facts with the same coordinates. Finally, Claim 4 restricts term equalities to the above cases.

Example 2. Consider a DS $\mathcal{D}$ containing the four types a, b, c, d in Figure 2. In the simulation mapping $\Sigma_{\mathcal{D}}$, the first type is modelled by the following implicational conjunct in the SO tgd τ_a: $C_h(x_1, x_2) \wedge C_v(y_1, y_2) \rightarrow D(w^h\, x_1\, y_2, w^v\, x_2\, y_1, g^h\, x_2\, y_2, g^v\, x_2\, y_2)$. Here, w and g stand for white and gray colours, respectively.

The 3×2 grid is represented by a proper source instance I (see top of Figure 2). Six dotted rectangles represent a consistent tiling of this grid with $\mathcal{D}$. Each rectangle bears the four terms of the corresponding D-fact in $chase(I, \Sigma_\mathcal{D})$: E.g., the topmost leftmost tile (the one at position (1,1), that is) is encoded by the fact $D(w^h01, w^v10, g^h11, g^v11)$.

Problem reduction. Let $\mathcal{D}$ be a DS and $\Sigma_\mathcal{D}$ its simulation mapping. Furthermore, consider a singleton colour set $\mathcal{C}_b = \{b\}$ (where b stands for "black") and a corresponding domino set $\mathcal{B} = \{\langle b, b, b, b\rangle\}$ with the simulation mapping $\Sigma_\mathcal{B}$. We claim that $\mathcal{D}$ has a solution iff $\Sigma_\mathcal{D} \equiv_{CQ} \Sigma_\mathcal{B}$ holds.

The SO tgd in $\Sigma_\mathcal{B}$ has the following form: $\tau_\mathcal{B}$: $\exists b^h \exists b^v \exists p^h \exists p^v \exists q^h \exists q^v \big(\big(C_h(x_1, x_2) \wedge C_v(y_1, y_2) \rightarrow D(b^h\, x_1\, y_2, b^v\, x_2\, y_1, b^h\, x_2\, y_2, b^v\, x_2\, y_2)\big) \wedge \gamma_h \wedge \gamma_v\big)$. It is easy to see that the conjuncts γ_h and γ_v are redundant in $\tau_\mathcal{B}$. Hence, in the following we assume $\tau_\mathcal{B}$ to contain a single implicational conjunct and no guards. Moreover, the following relationship holds between the chase result of $\tau_\mathcal{B}$ and $\tau_\mathcal{D}$:

Property 2. Let $\mathcal{D}$ be a DS with simulation mapping $\Sigma_\mathcal{D}$, let (i_1, j_1) and (i_2, j_2) be two distinct grid positions, and let $d_1, d_2 \in chase(I, \Sigma_\mathcal{D})$ be facts at positions (i_1, j_1) resp. (i_2, j_2). Then also $chase(I, \Sigma_\mathcal{B})$ contains two facts d'_1, d'_2 at these grid positions. Moreover, for every $1 \leq k, l \leq 4$, if $d_1.k = d_2.l$ then $d'_1.k = d'_2.l$, i.e., if two terms in d_1 and d_2 are equal, then the terms at these places in d'_1 and d'_2 are also equal.

This property uses an observation, that every implicational conjunct τ_{d_i} encoding a domino type d_i in the SO tgd of a simulation mapping produces exactly one D-fact for each grid position, and that $\tau_\mathcal{B} \in \Sigma_\mathcal{B}$ uses the minimal set of functional symbols.

For DPs that have a solution (of which $\mathcal{B}$ is obviously an example) the following lemma, which is crucial for our construction, holds:

Lemma 1. *Let I be a proper instance of size (m, n) and let $\mathcal{D}$ be a DS. Then, there is a consistent tiling of an $m \times n$ grid with $\mathcal{D}$ iff $core(I, \tau_\mathcal{D}) \cong core(I, \Sigma_\mathcal{B})$ holds.*

Example 3. It is easy to check, that the six D-facts in Figure 2 indeed form $core(I, \Sigma_\mathcal{D})$. For I, $\Sigma_\mathcal{B}$ yields a solution with the isomorphic core: each fact in $core(I, \Sigma_\mathcal{B})$ can be obtained from some fact of $core(I, \Sigma_\mathcal{D})$ by replacing leading symbols w and g with b.

To lift the equivalence between solvability of $\mathcal{D}$ and existence of an isomorphism $core(I, \Sigma_\mathcal{D}) \cong core(I, \Sigma_\mathcal{B})$ to unrestricted source instances I, we will first show, that whenever C_h^I or C_v^I contains a single cycle (that is, its dual graph is connected, each vertex having a single incoming and a single outgoing edge), then any simulation mapping yields an instance with the same core as $chase(I, \Sigma_\mathcal{B})$.

Lemma 2. *Let I contain a single cycle in C_h or in C_v. Then, for arbitrary DS $\mathcal{D}$, $core(I, \Sigma_\mathcal{D}) \cong core(I, \Sigma_\mathcal{B})$ holds.*

Proof (Sketch). W.l.o.g. assume that the single cycle is contained in C_h. For such I, the guard γ_h in the SO tgd $\tau_\mathcal{D} \in \Sigma_\mathcal{D}$ is indistinguishable from $\tau_\mathcal{B}$ in $\Sigma_\mathcal{B}$. Indeed, the conclusions of these implications coincide up to renaming of functional terms. The antecedent φ_h of γ_h has an extra atom $C(x_0, x_1)$, that prevents it from being satisfied by a C_h-fact at the beginning of the chain. However, if C_h^I is a cycle, the relations $q_{1|2}$

defined as $q_1(x_1, x_2) \leftrightarrow C_h(x_1, x_2)$ and $q_2(x_1, x_2) \leftrightarrow \exists x_0\ C_h(x_0, x_1) \wedge C_h(x_1, x_2)$, coincide, and so do $\tau_{\mathcal{B}}$ and γ_h. Hence, $J = chase(I, \Sigma_{\mathcal{D}})$ contains a subinstance $J_{\mathcal{B}}$ isomorphic to $chase(I, \Sigma_{\mathcal{B}})$. Using Property 2, $J \rightarrow J_{\mathcal{B}}$ can be shown. $\square$

Next, we address the source instances I containing multiple connected components in $C^I_{h|v}$. The *connectedness* is defined as follows: Given an instance K, the *fact graph* $\mathcal{G}_K$ has the facts of K as vertices, connected by an undirected edge whenever corresponding facts have a common term. The facts $f_1, f_2 \in K$ are connected, if there is a path between the corresponding vertices in $\mathcal{G}_K$. The following property can be shown by induction on the length of the path between two D-facts:

Lemma 3. *Let $\Sigma_{\mathcal{D}}$ be a simulation mapping, I a source instance over $\mathbf{S}$, and let d_1, d_2 be facts in $J = chase(I, \Sigma_{\mathcal{D}})$ containing respectively terms $\alpha(x_1, y_1)$ and $\beta(x_2, y_2)$. Assume that x_1 is a term in a fact $c'_h \in C^I_h$, and x_2 in $c''_h \in C^I_h$. Likewise, let y_1 and y_2 occur, respectively, in the facts $c'_v, c''_v \in C^I_v$. Then, d_1 and d_2 are connected in J only if the facts c'_h and c''_h are connected in C^I_h, and so are c'_v and c''_v in C^I_v.*

The above lemma implies, that if the source I contains l_h and l_v connected components in C^I_h and C^I_v respectively, then for any simulation mapping Σ, $chase(I, \Sigma)$ can be decomposed into $l_h l_v$ pairwise disconnected instances. Furthermore, if dual graphs of the connected components of $C^I_{h|v}$ are acyclic chains or cycles, then by applying one of the Lemmas 1, 2 in each of the $l_h l_v$ cases, the reduction established by these lemmas can be lifted to the the case when relations $C^I_{h|v}$ contain multiple connected components.

Finally, we observe that union of chains and cycles is the only type of dual graphs which is relevant. Indeed, if the dual graph of a source instance I contains a V-formed subgraph $\{C(x, y_1), C(x, y_2)\}$ or $\{C(y_1, x), C(y_2, x)\}$ with $y_1 \neq y_2$, then the key dependencies ensure that for any simulation mapping Σ (including $\Sigma_{\mathcal{B}}$) the set $Sol(I, \Sigma)$ is empty. This concludes our reduction from the domino problem to CQ-equivalence of SO tgds. We may thus formulate the main result of this section.

Theorem 3. *The CQ-equivalence of mappings consisting of SO tgds and source key dependencies is undecidable.*

For the simulation mappings $\Sigma_{\mathcal{D}}$ and $\Sigma_{\mathcal{B}}$ in the above problem reduction, it can be easily proved that $\Sigma_{\mathcal{D}} \equiv_{CQ} \Sigma_{\mathcal{B}}$ holds iff $\Sigma_{\mathcal{D}} \equiv \Sigma_{\mathcal{B}}$ holds. We thus immediately get

Theorem 4. *The logical equivalence of mappings consisting of SO tgds without equalities in the antecedents and source key dependencies is undecidable.*

5 Conclusion

Testing the equivalence of schema mappings is a fundamental task. For mappings defined by s-t tgds and target dependencies in the form of tgds and equality generating dependencies (egds) this problem has been well studied. In the general case, the (logical) equivalence of such mappings is undecidable. If the target tgds admit a finite chase, then the problem becomes decidable [4]. In [10], two relaxed notions of equivalence were introduced, namely the CQ-equivalence studied here and data exchange equivalence

(DE-equivalence). Moreover, in [10], the undecidability of CQ-equivalence was proved for mappings consisting of s-t tgds and target tgds. The undecidability was extended in [17] in two directions, namely to mappings with target egds and from CQ-equivalence to DE-equivalence. The equivalence of SO tgds has been unexplored so far.

In this paper, we have proved the undecidability of logical equivalence. For the case of source schemas with key dependencies, we proved the undecidability of CQ-equivalence. The status of CQ-equivalence without any assumptions on the source schema has been left open. Closing this gap is on top of the agenda for future work. Moreover, the search for decidable fragments should be initiated. Finally, we also want to study the DE-equivalence [10] mentioned above. We conjecture that DE-equivalence of SO tgds is also undecidable. However, this has to be proved yet.

References

1. C. Allauzen and B. Durand. *The classical decision problem, by E. Börger, E. Grädel and Yu. Gurevich*, chapter "Appendix A: Tiling Problems", pages 407–420. Springer, 2001.
2. M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Composition and inversion of schema mappings. *SIGMOD Record*, 38(3):17–28, 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. R. Berger. *The undecidability of the domino problem*. Amer Mathematical Society, 1966.
6. P. A. Bernstein. Applying model management to classical meta data problems. In *Proc. CIDR'03*, 2003. available from http://www-db.cs.wisc.edu/cidr/cidr2003/program/p19.pdf.
7. P. A. Bernstein, T. J. Green, S. Melnik, and A. Nash. Implementing mapping composition. *VLDB J.*, 17(2):333–353, 2008.
8. R. Fagin. Inverting schema mappings. In *Proc. PODS'06*, pages 50–59. ACM, 2006.
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. In *Proc. PODS'08*, pages 33–42. ACM, 2008.
11. R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM Trans. Database 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. A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data integration: The teenage years. In *Proc. VLDB'06*, pages 9–16. ACM, 2006.
14. J. Madhavan and A. Y. Halevy. Composing mappings among data sources. In *Proc. VLDB'03*, pages 572–583, 2003.
15. S. Melnik. *Generic Model Management: Concepts and Algorithms*, volume 2967 of *Lecture Notes in Computer Science*. Springer, 2004.
16. A. Nash, P. A. Bernstein, and S. Melnik. Composition of mappings given by embedded dependencies. *ACM Trans. Database Syst.*, 32(1):4, 2007.
17. R. Pichler, E. Sallinger, and V. Savenkov. Relaxed notions of schema mapping equivalence revisited. In *Proc. ICDT'11*, pages 90–101. ACM, 2011.