Value Invention in Data Exchange

Patricia C. Arocena
University of Toronto
prg@cs.toronto.edu

Boris Glavic
Illinois Institute of Technology
bglavic@iit.edu

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

ABSTRACT

The creation of values to represent incomplete information, often referred to as *value invention*, is central in data exchange. Within schema mappings, Skolem functions have long been used for value invention as they permit a precise representation of missing information. Recent work on a powerful mapping language called *second-order tuple generating dependencies* (SO tgds), has drawn attention to the fact that the use of arbitrary Skolem functions can have negative computational and programmatic properties in data exchange. In this paper, we present two techniques for understanding when the Skolem functions needed to represent the correct semantics of incomplete information are computationally well-behaved. Specifically, we consider when the Skolem functions in second-order (SO) mappings have a first-order (FO) semantics and are therefore programmatically and computationally more desirable for use in practice. Our first technique, *linearization*, significantly extends the Nash, Bernstein and Melnik *unskolemization* algorithm, by understanding when the sets of arguments of the Skolem functions in a mapping are related by set inclusion. We show that such a linear relationship leads to mappings that have FO semantics and are expressible in popular mapping languages including source-to-target tgds and nested tgds. Our second technique uses source semantics, specifically functional dependencies (including keys), to transform SO mappings into equivalent FO mappings. We show that our algorithms are applicable to a strictly larger class of mappings than previous approaches, but more importantly we present an extensive experimental evaluation that quantifies this difference (about 78% improvement) over an extensive schema mapping benchmark and illustrates the applicability of our results on real mappings.

Categories and Subject Descriptors

H.2.5 [**Heterogeneous Databases**]: Data Translation

Keywords

Data Exchange, Schema Mappings, Value Invention, Unskolemization

1. INTRODUCTION

Schema mappings are constraints that model the relationship between schemas and have been used extensively in data exchange and integration. In data exchange, schema mappings are used as a logical specification of how to map data from a source to a target schema. In data integration, mappings are used to specify how target queries can be translated into source queries. For decades, Skolem functions have played an important role in schema mappings [18]. Skolem functions can be used to model object identifier (OID) invention or *value invention* [18], and more generally to provide a precise model of how missing values are correlated to known source values [27, 29], and to other missing values [1]. The representational power of functions in mappings was recognized by Fagin et al. [12], with their presentation of a class of existential second-order (SO) formulas with arbitrary function symbols, called *second-order tuple-generating dependencies* (SO tgds). SO tgds are the right choice for modeling many data exchange and integration problems, such as mapping composition [12] and correlation of mappings [1]. Some, but not all, SO tgds are equivalent to *first-order (FO)* mappings (that is, mappings expressible in FO logic) [12]. While having the right expressive power, SO tgds do have some drawbacks. Model checking, determining whether a pair of instances satisfies a mapping, is intractable for SO tgds, while polynomial in data complexity for FO mappings [12]. Model checking is important for supporting updates, peer data-exchange [15], and data coordination [20]. Furthermore, SO tgds are not closed under target homomorphisms [31], a property that has undesirable consequences in practice. If a data exchange solution (for an SO tgd) is modified by a homomorphism that preserves constants and only modifies `NULLs` (for example by filling in missing information), then the result may no longer be a valid solution.

As a result, most systems, including Clio [29], Orchestra [17], ++Spicy [24], and OpenII [30], use FO mapping languages based on the popular language of *source-to-target (s-t)* tgds (also known as Global-and-Local-As-View or *GLAV* for short [21]), and the more general *nested* tgds (*nested GLAV*) [14]. Fuxman et al. [14] argued that nested GLAV, which permits nesting of mappings, instead of correlation of mappings via Skolem functions (as in SO tgds), are a less powerful, but more user-friendly programming concept. Both GLAV and nested GLAV mappings are closed under target homomorphisms [31]. Translations of these FO mapping languages into efficient transformation code (in, e.g.,

Source Schema

WorksOn (<u>Department, Project</u>, BudgetId)
Audit (<u>BudgetId</u>, Auditor)
City (Department, City)

Target Schema

Project (PId, BudgetId)
Dept (Department, Year, Project, NumEmp)
Location (Department, DepId, City, State)
Budget (Project, Leader, Size)

Figure 1: Example Schema

SQL, XQuery, or XSLT) are well-known, [14, 24] and others, but the same is not true for arbitrary SO tgds.

Fortunately, many (though of course not all) SO tgds are equivalent to (nested) GLAV mappings. However, testing if an SO tgd is equivalent to a (nested) GLAV mapping is undecidable, so it is not possible to have a complete characterization of all SO tgds that are rewritable (this follows from the result of Feinerer et al. [13]). Nash et al. [26] were the first to present a polynomial-time sufficient condition for when an SO tgd is equivalent to a GLAV mapping and to motivate why finding such rewritings can be important in practice. They provided a transformation algorithm to map SO tgds produced by mapping composition to an equivalent GLAV mapping, which can be of exponential size. We generalize their work to understand when the potentially correlated functions in an arbitrary SO tgd are well-behaved. We provide a more general condition that is able to transform more SO tgds to GLAV. We also consider transformations to the more expressive language of nested GLAV. The size of a nested GLAV mapping produced by these transformations is linear in the size of the input SO tgd.[1] We call SO tgds that can be transformed by our techniques *linearizable*. Finally, we consider source semantics, specifically functional dependencies (FDs). We show that in the presence of FDs (including keys), a larger set of SO tgds can be transformed to equivalent nested GLAV mappings.

1.1 Nesting and Linearization

The language of SO tgds is the right language for model management operators [9] like composition [12] and MapMerge [1], and has found important application in other operators including mapping inversion [6]. To motivate our work, we consider through examples, when such mappings can be simplified and expressed as a set of equivalent FO mappings (either GLAV or nested GLAV). Consider the source and target schemas of Figure 1. Primary key attributes are underlined.

EXAMPLE 1.1. *Suppose we wish to map the source information* `WorksOn.BudgetId` *to the* `Project` *relation in the target. The project id* (`PId`) *of the target is not present in the source, so a mapping must* invent *this value using an existential or Skolem function. One mapping (θ_1) that expresses the semantics that the target attribute* `PId` *should depend on the source values* `Department` *(d) and* `Project` *(p), is shown in Figure 2. This mapping is an SO tgd as it contains an existentially quantified function. Intuitively, this function models that any tuples with the same* `Department` *and* `Project` *values must necessarily share the same* `PId` *value. It is well known that this mapping is equivalent to the FO mapping Ω_1 (a nested GLAV mapping which permits interleaving of universal and existential quantifiers [14]). The position of the existential variable in Ω_1 (nested after d and p) expresses the same semantics as the existential function in θ_1. Fagin et al. [12] proved that this mapping is not equivalent to* **any** *GLAV mapping. As a result, the only known algorithm for simplifying SO tgds [26] would fail as it only considers transformation into GLAV mappings, not nested GLAV.*

In our work, we present a sufficient condition for simplifying SO tgds into nested GLAV mappings that would produce Ω_1. Intuitively, our condition characterizes when Skolem functions are well-behaved in that they do not interact with other Skolem functions in a way that would prohibit a simple FO substitution of a Skolem function with an appropriately nested existential. Of course, in the above example there is a single Skolem function so the replacement of the Skolem function by an existential only requires examination of its arguments. When there are multiple Skolem functions, we must understand how their arguments interact.

EXAMPLE 1.2. *Consider the second mapping in Figure 2 adapted from Libkin and Sirangelo [22], which maps the* `WorksOn` *relation to a target* `Dept` *relation. This example illustrates a mapping in which the arguments to the Skolem functions have been* customized *to precisely model the application semantics. In the SO tgd θ_2, the two Skolem functions f and g are used to generate values for the* `Year` *and* `NumEmp` *attributes. The target attribute* `Year` *depends on the department and, thus, d* (`Department`)*, is the only argument for Skolem function f. The Skolem function g has two arguments, because the number of employees* (`NumEmp`) *depends on the* `Department` *and* `Project`*. Again, this SO tgd is not equivalent to any GLAV mapping [22]. Our algorithm succeeds by placing the (FO) existentials in the correct scope, and would produce Ω_2, an equivalent nested GLAV mapping.*

Notice that to produce an equivalent nested mapping, the Skolem functions in the SO tgd (and corresponding existentials) must be placed in a **linear** order. This example adapts an existing SO quantifier elimination method, often called *linearization* to the case of SO tgds. We are not the first to make use of this technique for data exchange. Pichler and Skritek [28] proposed a fragment of SO tgds named SO^{ord} for which the complexity of model checking, both data and combined complexity, can be reduced to that of FO formulas. In their reduction, the authors impose conditions that rely on the existence of a total ordering of the Skolem functions (based on set containment of their argument sets). Our rewriting techniques will generalize this by using a novel partitioning strategy that allows us to apply linearization over sets of clauses in a complex SO tgd. When these clauses can be partitioned into sets (which we call blocks) with pairwise disjoint sets of (linearizable) Skolem functions, our condition is more general. Furthermore, unlike Pichler and Skritek [28], we present a constructive (and efficient) algorithm for translating this more general set of SO tgds into nested GLAV mappings.

1.2 (Un)Skolemization Revisited

Our examples illustrate that we will be replacing Skolem functions with existentials, a process known under various

[1]This does not contradict the result from Nash et al. [26] stating that GLAV rewritings can be exponential in size, because we are rewriting into nested GLAV mappings which are more expressive.

Ex. 1: Key Invention

$\theta_1 = \exists f\,(\forall d \forall p \forall b\, W(d,p,b) \rightarrow P(f(d,p),b))$

$\Omega_1 = \forall d \forall p\, \exists v_f\, \forall b\, W(d,p,b) \rightarrow P(v_f,b)$

Ex. 2: Skolems with Overlapping Arguments

$\theta_2 = \exists f \exists g\,(\forall d \forall p \forall b\, W(d,p,b) \rightarrow D(d,f(d),p,g(d,p)))$

$\Omega_2 = \forall d\, \exists v_f\, \forall p\, \exists v_g\, \forall b\, W(d,p,b) \rightarrow D(d,v_f,p,v_g)$

Ex. 3.A: Skolems with Disjoint Arguments

$\theta_3 = \exists f \exists g\,(\forall d \forall c\, C(d,c) \rightarrow L(d,f(d),c,g(c)))$

Ex. 3.B: FD on City: Department → City

$\theta_3' = \exists f \exists g\,(\forall d \forall c\, C(d,c) \rightarrow L(d,f(d,c),c,g(c)))$

$\Omega_3 = \forall c\, \exists v_g\, \forall d\, \exists v_f\; C(d,c) \rightarrow L(d,v_f,c,v_g)$

Ex. 4: Arguments from Distinct Source Relations

$\theta_4 = \exists f \exists g\,(\forall d \forall p \forall b \forall a\, W(d,p,b) \wedge A(b,a) \rightarrow B(p,f(d,p,b),g(b,a)))$

$\Omega_4 = \forall b \forall a\, \exists v_g\, \forall d \forall p\, \exists v_f\; W(d,p,b) \wedge A(b,a) \rightarrow B(p,v_f,v_g)$

WorksOn FD: Department, Project → BudgetId

Audit FD: BudgetId → Auditor

Figure 2: SO tgds and First-order Translations

names such as *unskolemization*, *reversed Skolemization*, and *deskolemization* [16, 26]. Important to our solutions will be determining the right placement of the existentials (nested within universal quantifiers) based on the arguments of the Skolem functions and also whether such a placement is possible. As our examples make clear, the existential quantifier must be placed after any universal quantifiers representing its arguments (and not after universals on which it does not depend). To understand this, it is helpful to consider both unskolemization and its complement Skolemization. Indeed, in the integration literature, Skolemization is often used as a process for transforming FO formulas into a normal form that *eliminates* existential quantifiers by replacing them with functions. The reverse process (unskolemization) is always possible for Skolemized FO formulas, but not for SO tgds because SO tgds can have arbitrary arguments in the Skolem functions. Our work considers sufficient conditions for when this is possible. So while our algorithms can be considered unskolemization algorithms, it is important to recognize that our goal is not to get rid of Skolem functions, but rather to understand when the Skolem functions in a mapping are well-behaved (and have a computationally desirable FO semantics). If a complex SO tgd can be shown to be equivalent to a (nested) GLAV mapping then we guarantee that the Skolem functions can be untwisted to produce a well-behaved nested mapping, and therefore efficient transformation code.

Example 1.3. (Part A) *Consider the mapping θ_3 in Figure 2 that maps the source relation* `City` *to the target relation* `Location`*. The target relation has two attributes,* `DepId` *and* `State`*, which depend respectively on the* `Department` *and* `City` *attributes from the source. This SO tgd is* **not** *equivalent to Ω_3 or to any nested GLAV mapping [12]. Thus, for example, if the source and target evolve independently of each other it may be computationally hard to check whether the modified instances fulfill the mapping, because this is model checking which is NP-complete for SO tgds (and in P for nested GLAV mappings).*

1.3 Source Constraints

Often the source contains constraints, most commonly keys or FDs. We can use these constraints to simplify SO mappings that otherwise would not be equivalent to any nested mapping (like θ_3).

Example 1.3.(Part B) *Suppose each* `Department` *in the source has a single* `City` *(modeled by the FD* `Department` → `City`*). If this is the case, then we will show that θ_3 is equivalent to θ_3' (Figure 2) where the Skolem f has arguments d and c (both* `Department` *and* `City`*). This enables us to use our linearization criteria to show that θ_3 with this FD, is equivalent to the nested mapping Ω_3.*

Source constraints are a powerful tool for simplifying mappings, but we can do even better by considering implied dependencies. If we consider a source expression as a view, the set of FDs implied by the view can be computed [19]. The next example illustrates how to use such inferred constraints to simplify SO tgds.

Example 1.4. *Consider the SO tgd θ_4 which maps a join of* `WorksOn` *and* `Audit` *to the target relation* `Budget` *(adapted from Alexe et al. [1]). Suppose we have source keys (*`Department, Project`*) for* `WorksOn` *and* `BudgetId` *for* `Audit`*. These keys alone cannot be used to augment the arguments of Skolem functions f or g (since both already contain the keys). However, if we compute the implied dependencies on the source join, we will see that $d, p \rightarrow b, a$ (over the exchanged data). Hence, we can augment the arguments of f to include all four variables and easily see that θ_4 with these keys is equivalent to Ω_4 which linearizes g before f.*

1.4 Summary of Contributions

The main contributions of this work are the following.

- We prove a sufficient condition for an SO tgd to be equivalent to a nested tgd. This condition is broader than the condition used in previous work by Nash et al. [26], and we experimentally compare this difference.
- We develop an algorithm (called `Linearize`) for rewriting SO tgds satisfying our condition into nested GLAV mappings. Our algorithm is more general in that it produces nested GLAV mappings, uses linearization of Skolem arguments, and includes a new partitioning scheme for complex, multi-clause SO tgds.
- We introduce an equivalence preserving transformation of SO tgds exploiting the existence of FDs in the source schema. Our transformation computes attribute closures for the source expression considered as a view, and uses these to augment the arguments of Skolem functions.
- We present an algorithm `LinearizeFDs` that extends our linearization technique with source FDs, and show that this combination presents many more opportunities for rewriting. Specifically, many SO tgds that cannot be rewritten, can be written in the presence of source keys or FDs. We compare this against Marnette et al.'s re-Skolemization strategy [23], originally proposed to compute compact solutions.
- We use STBenchmark 2.0 [7] to produce SO tgds and mapping scenarios with source keys or FDs. We use this extended system to generate a large set of mapping scenarios that include schemas produced by composition, evolu-

tion, or manual customization of Skolem functions. We also present real-life mapping scenarios. Our evaluation shows that (1) our partitioning and linearization techniques permit the rewriting of significantly more SO tgds than previous techniques (approximately 78% more), (2) by using source keys (that is, normalized source schemas where the only FDs are keys) we can rewrite over twice as many SO tgds as without any keys.

The remainder of the paper is organized as follows. We introduce some formal definitions and background in Section 2. Section 3 presents our new partitioning and linearization conditions followed by our rewriting algorithm `Linearize` in Section 4. Building on these results, Section 5 demonstrates how to use FDs to provide more opportunities for rewriting and presents our `LinearizeFDs` algorithm. We empirically evaluate our techniques in Section 6, discuss additional related work in Section 7, and conclude in Section 8.

2. PRELIMINARIES

Schemas and Instances. A *schema* is a non-empty finite set $\mathbf{R}$ of relation symbols where each $R_i \in \mathbf{R}$ has a fixed arity. We define the notion of *instance* I over a schema $\mathbf{R}$ in the normal way for relational schemas, that is, as the union of relation instances over R_i, where $R_i \in \mathbf{R}$. Let $\mathbf{S} = (S_1, \ldots, S_n)$ and $\mathbf{T} = (T_1, \ldots, T_m)$ be two disjoint schemas. We follow the convention of referring to $\mathbf{S}$ as the *source* schema and to $\mathbf{T}$ as the *target* schema. The notation $(\mathbf{S}, \mathbf{T})$ denotes the schema $(S_1, \ldots, S_n, T_1, \ldots, T_m)$. When necessary, we extend this nomenclature to instances by adding the prefix *source* and *target*, respectively.

Satisfaction, Quantifier Scope, and Substitution. We use the standard notion of satisfaction of a formula in FO logic. If K is an instance and φ is a formula, we write $K \models \varphi$ to denote that K satisfies φ. The same notion is applied over a set of formulas Σ. The scope of an occurrence of a quantifier $Q \in \{\forall, \exists\}$ in a formula is the subformula controlled by the quantifier. For example, given $\forall x \exists y \psi(x, y)$, the scope of quantifier $\forall x$ is the subformula $\exists y \phi(x, y)$, where x appears as a free variable, and the scope of $\exists y$ is the subformula $\phi(x, y)$, where both x and y are free. Let ϕ be a formula. We use $\phi[t \leftarrow t']$ to denote the formula that is derived from ϕ by replacing all occurrences of term t with term t'.

Schema Mappings. A *schema mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\mathbf{S}$ and $\mathbf{T}$ are schemas with no relation symbols in common and Σ is a set of logical formulas over $(\mathbf{S}, \mathbf{T})$ [11]. An *instance of* $\mathcal{M}$ is an instance $\langle I, J \rangle$ over $(\mathbf{S}, \mathbf{T})$ that satisfies every formula in Σ. If $\langle I, J \rangle \models \mathcal{M}$, then we call J a *solution of* I *under* $\mathcal{M}$.

First-Order Tgds. A *source-to-target tuple-generating dependency* (s-t tgd) [11] is a formula of the form $\forall \mathbf{z}, \mathbf{x}(\phi(\mathbf{z}, \mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$, where $\mathbf{z}$, $\mathbf{x}$, and $\mathbf{y}$ are disjoint vectors of variables; $\phi(\mathbf{z}, \mathbf{x})$ is a conjunction of atomic formulas over the source schema $\mathbf{S}$; and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over the target schema $\mathbf{T}$. All variables of $\mathbf{z} \cup \mathbf{x}$ are used in ϕ and all variables of $\mathbf{x} \cup \mathbf{y}$ are used in ψ. We adopt the term **GLAV mappings** [21] for sets of s-t tgds. A *nested tgd* [14] is an FO sentence of the form $Q(\mathbf{x}, \mathbf{y})((\phi_1(\mathbf{x}) \rightarrow \psi_1(\mathbf{x}, \mathbf{y})) \wedge \ldots \wedge (\phi_n(\mathbf{x}) \rightarrow \psi_n(\mathbf{x}, \mathbf{y})))$ where (1) $Q(\mathbf{x}, \mathbf{y})$ is a sequence of quantifiers - universal quantifiers for $\mathbf{x}$ and existential quantifiers for $\mathbf{y}$; (2) each $\phi_i(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over the source schema $\mathbf{S}$; (3) each $\psi_i(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over the target schema $\mathbf{T}$; and (4) each variable from $\mathbf{x}$ appears in some formula ψ_i and each variable from $\mathbf{y}$ appears in some formula ψ_j. Nested tgds, also known as **nested GLAV mappings**, are a proper extension of GLAV mappings that allows alternation of $\forall$ and $\exists$ quantifiers [31].

Second-Order Tgds. A *second-order tuple-generating dependency* (SO tgd) [12] is an existential second-order formula of the form $\exists \mathbf{f}((\forall \mathbf{x_1}(\phi_1 \rightarrow \psi_1)) \wedge \cdots \wedge (\forall \mathbf{x_n}(\phi_n \rightarrow \psi_n)))$ where (1) each member of $\mathbf{f}$ is a function symbol; (2) each ϕ_i is a conjunction of (a) atomic formulas over the source schema $\mathbf{S}$ and (b) equalities of the form $t = t'$ where t and t' are terms based on $\mathbf{x_i}$ and $\mathbf{f}$; (3) each ψ_i is a conjunction of atomic formulas over the target schema $\mathbf{T}$; and (4) each variable in $\mathbf{x_i}$ appears in some atom of ϕ_i. Even though SO tgds may allow equalities between or with Skolem terms in general settings, such equalities do not play a role for data exchange and can be eliminated [32]. Arenas et al. [5] have generalized previous results from Nash et al. [26] to prove that every SO tgd is equivalent to another SO tgd with no nested Skolem functions (e.g., $f(g(a), b)$). Thus, in what follows, we assume SO tgds without equalities between or with Skolem terms and no nesting of Skolem terms.

Clausal Normal Form. Any SO tgd can be transformed into an equivalent SO tgd in *clausal normal form*. Let $\exists \mathbf{f}\, \forall \mathbf{x_1} \ldots \forall \mathbf{x_n} (\chi_1 \wedge \cdots \wedge \chi_n)$ be an SO tgd θ, where each implication clause χ_i is a quantifier-free formula (i.e., θ is in prenex normal form). Let $N = \{C_1, \ldots, C_n\}$ be a set of clauses where $C_i = \chi_i$ for $1 \leq i \leq n$. We refer to N as the clausal normal form of θ where the variables in N are implicitly universally quantified, and the clauses are related by conjunction [16]. Unless otherwise stated, we assume SO tgds in clausal normal form.

Skolemization. Let α be an FO formula. The SO *Skolemization* of α is an SO formula that results from replacing existential quantifiers with Skolem functions, based on the equivalence $\forall \mathbf{x} \exists y\, \delta(\mathbf{x}, y) \equiv \exists f \forall \mathbf{x}\, \delta(\mathbf{x}, y)[y \leftarrow f(\mathbf{x})]$ [10]. Informally, this transformation removes $\exists y$ from α, introduces a new existentially quantified *Skolem function* f of *arity* k, where k is the number of universally quantified variables in $\mathbf{x}$, and replaces y wherever it appears in the scope of $\exists y$ with the Skolem term $f(\mathbf{x})$. The *argument sequence* of a Skolem function $f(x_1, x_2, \ldots, x_k)$, denoted as $arg(f)$, is the sequence of variables $x_1, x_2, \ldots, x_k$ on which f depends. We use the term *argument set* (and abuse the notation $arg(f)$) to refer to the set of variables in the argument sequence.

3. SUFFICIENT REWRITING CONDITION

We now present the first contribution of the paper, a *new* sufficient condition for rewriting SO tgds into nested GLAV mappings. This condition relies on the notions of *consistency* and *linearity*, which we adapt from SO quantifier elimination methods [16]. We first present the intuition, then prove the correctness of our condition. Importantly, we turn this condition into a practical rewrite algorithm in Section 4.

3.1 Intuition of Unskolemization

Our approach for rewriting SO tgds relies heavily on unskolemization. Consider an SO tgd θ in prenex form [10]

$$\exists \mathbf{f}\, \forall \mathbf{x_1} \ldots \forall \mathbf{x_n} (\chi_1 \wedge \cdots \wedge \chi_n) \tag{1}$$

The aim of unskolemization is to eliminate the $\exists \mathbf{f}$ quantifiers from θ and produce an equivalent set (Ω in our examples) of FO formulas, in our case a nested GLAV mapping, without Skolem functions. This involves systematically introducing

FO existential quantifiers in place of Skolem functions. To do this, we resort to the same logical equivalence transformation [10] used to Skolemize FO formulas, i.e.,

$$\forall \mathbf{x} \exists y \; \delta(\mathbf{x}, y) \equiv \exists f \forall \mathbf{x} \; \delta(\mathbf{x}, y)[y \leftarrow f(\mathbf{x})] \qquad (2)$$

For unskolemization, we aim to apply Eq. (2) from right-to-left. For instance, reversing the original Skolemization of an FO formula involves repeated applications of this single-step equivalence over each Skolem function symbol. Notice that when doing this, we implicitly obey an equation's predetermined order of universal and existential quantifiers (some of which may have been introduced by previous applications of the equivalence). That is, for $f(\mathbf{x})$, we must add $\exists y$ immediately after all universal quantifiers binding $\mathbf{x}$.

Example 3.1. *Consider a single clause SO tgd:*

$$\exists f \exists g \; (\forall x_1 \forall x_2 (R(x_1, x_2) \rightarrow S(f(x_1), g(x_1, x_2), x_2)))$$

To transform this SO tgd into an equivalent nested GLAV mapping, we apply Eq. (2) twice. First, we replace g with an existential variable v_g that is introduced right after the quantification of x_1 and x_2, the arguments of g. The resulting SO formula θ' is $\exists f (\forall x_1 \forall x_2 \exists v_g \; (R(x_1, x_2) \rightarrow S(f(x_1), v_g, x_2)))$. We can now apply Eq. (2) again to remove the function f. An existential variable v_f representing f can be introduced right after the quantifier for x_1 (the argument of f). The resulting equivalent nested GLAV mapping is:

$$\forall x_1 \exists v_f \forall x_2 \exists v_g (R(x_1, x_2) \rightarrow S(v_f, v_g, x_2))$$

In this example, the first application of Eq. (2) (to remove g) did not hinder us from applying the equation a second time to remove the function f. However, when reversing the Skolemization of an arbitrary SO tgd, this may not always be the case, i.e., a rewriting step may result in a formula that can not be further unskolemized using this equation. This is not surprising, because there are SO tgds that are not equivalent to any FO formula. The conditions over SO tgds we introduce next make use of a careful ordering of unskolemization steps to ensure that this type of rewriting is indeed possible, i.e., they are *sufficient*.

3.2 Consistency

The first condition we consider is *consistency*. Consistency models whether all occurrences of a Skolem function symbol in a formula use exactly the same arguments.

Definition 3.2. **[Consistency]** *A Skolem function f is consistent with respect to an SO tgd θ if 1) any two occurrences of f in θ have the same argument sequence, and 2) for any occurrence of f, each argument position is filled with a universally quantified source variable and no variables are repeated. An SO tgd θ is consistent when all its Skolem functions are consistent.*

By requiring each argument position to be filled with a universally quantified source variable, our definition disallows nesting of Skolem functions. At first sight, this may appear restrictive, but recall from Section 2 that any SO tgd is equivalent to an SO tgd without nesting of functions.

Example 3.3. *Consider the following set of clauses:*

$$C_1 : S(x_1, x_2) \rightarrow T_1(x_1, f(x_1), g(x_1, x_2)) \wedge T_2(g(x_2, x_1))$$
$$C_2 : R(x_3, x_4) \rightarrow W(x_3, x_4, h(x_3, x_3))$$

The function f is consistent, g is not (because the two occurrences of g do not share the same argument sequence), and h is not (it contains a repeated variable).

Lack of consistency is problematic when we try to unskolemize a formula because it may prevent us from choosing the same existential variable $\exists v_f$ to replace all occurrences of a given Skolem function f. Unification can often be applied to achieve consistency [16] but this is orthogonal to our discussion. Importantly, consistency has also been used by Nash et al. [26] (though using a different condition).

3.3 Maximal Partition

Consistency on its own does not guarantee rewritability. The linearity condition we motivated in Section 1 (based on conditions used in SO logic [16]) with consistency is still not sufficient for SO tgds. To apply linearity and consistency, we must first consider how the multiple clauses of an SO tgd are related. We present a new partitioning strategy for SO tgds (in which all clauses using a Skolem function f must be grouped together) that we use to guide rewriting.

Definition 3.4. **[Maximal Partition]** *Let θ be an SO tgd and $\Pi = \{\pi_1, \ldots, \pi_b\}$ a partition of the clauses of θ into sets of disjoint clauses called blocks. A partitioning Π is called a* maximal partition *of θ if (1) every two distinct blocks π_i and π_j are pairwise disjoint with respect to Skolem functions and (2) there exists no partition Π' of θ satisfying (1) that has more elements than Π.*

Note that every SO tgd θ has a unique maximal partition. For example, the maximal partition of an SO tgd with clauses $C_1 : R(x_1) \rightarrow S(f(x_1), g(x_1))$, $C_2 : T(x_1) \rightarrow U(g(x_1))$, and $C_3 : V(x_3) \rightarrow W(h(x_3))$ is $\{C_1, C_2\}, \{C_3\}$. This method of splitting an SO tgd into a conjunction of SO tgds based on the blocks in its maximal partition is equivalence preserving, because we split the input using the rules of quantifier distribution and SO tgds are closed under conjunction [12]. Thus, each block can be unskolemized separately and it is sufficient to check the linearity condition introduced next for each block individually. This is a less strict condition than requiring it to hold for the complete SO tgd as used in Pichler and Shritek [28].

3.4 Linearity

Linearity requires the Skolem functions of each block in the maximal partition Π of an SO tgd θ to form a linear order according to the set inclusion of their arguments.

Definition 3.5. **[Linearity]** *Let θ be an SO tgd and Π its maximal partition. A block π in Π is linear if for all distinct Skolem functions f and g occurring in π (with arity n and m, respectively) their argument sets can be linearly ordered by set inclusion. Assuming $n \leq m$, we have that the set of arguments of f is a subset of the set of arguments of g, i.e., $arg(f) \subseteq arg(g)$. We say an SO tgd θ is linear if all blocks π in Π are linear.*

Example 3.6. *Consider an SO tgd with a single block that contains Skolem terms $f(x_1, x_2)$, $g(x_1)$ and $h(x_3)$. While the argument sets of Skolem functions f and g can be linearly ordered by set inclusion (that is, $arg(g) \subseteq arg(f)$), notice that there is no set inclusion relationship between f and h, and between g and h. This block of clauses is not linear. However, if Skolem h would have been in a different block than f and g, then the SO tgd would be linear.*

3.5 Sufficient Condition

Linearity on top of consistency guarantees that we can produce an FO rewriting, where all the quantifier dependencies implied by Skolem function arguments can be morphed into a single linear quantifier prefix (as required to generate nested GLAV). It guarantees that when more than one existential variable needs to be introduced to replace the Skolem functions, Eq. 2 can be repeatedly applied, i.e., the result of each application of this equation is structured as required for the next application. More precisely, the equation should be applied to each Skolem function symbol in order of decreasing arity. For example, the SO tgd from Ex. 3.1 is linear ($f(x_1) < g(x_1, x_2)$) and, as has been shown in that example, can be transformed successfully into nested GLAV by replacing *first* Skolem function g and *then* f.

THEOREM 3.7. *Let θ be an SO tgd without equalities between or with Skolem terms. If θ is both consistent and linear, then θ can be successfully rewritten as a logically equivalent nested GLAV mapping.*

4. LINEARIZATION ALGORITHM

We now present a polynomial-time algorithm, called `Linearize`, that rewrites an SO tgd using our new sufficient condition from Section 3, and the right-to-left direction of the equivalence shown in Eq. (2). Algorithm 1 takes as input an SO tgd θ in clausal normal form. It attempts to rewrite the input by applying a number of transformation steps.

Check Consistency. Compute the maximal partition and test consistency. If any Skolem function is inconsistent, report that the rewriting failed.

Check Linearity. Recall from Example 3.6 that linearity can be checked on each block π of the maximal partition separately. We sort the list O_π of Skolem functions of block π according to set inclusion of their arguments using procedure `SortArgIncl`. If two Skolem functions are incomparable according to argument set inclusion, we report that the rewriting failed.

Rewrite into Nested GLAV. If the linearity condition is satisfied, rewrite each block of clauses π as a single nested tgd, yielding a nested GLAV mapping Ω. The algorithm creates a total order over the universally quantified variables based on the sorted order O_π of Skolem functions. Proceeding from lowest to highest function in sort order, for each Skolem term f, we add $\forall x$ to the mapping (for all $x \in arg(f)$ that are not already in the mapping due to an earlier Skolem term, notice that the ordering among these does not matter), then we add an existential variable for f, $\exists v_f$ (directly after the quantifiers for variables from the arguments of f). Once we have processed all variables used as arguments for any Skolem, we add an universal quantification $\forall x'$ for each remaining variable x'.

THEOREM 4.1. *Let θ be an SO tgd without equalities between or with Skolem terms, then `Linearize`$(\theta) \equiv \theta$.*

To illustrate Algorithm 1 consider our example SO tgds.

EXAMPLE 4.2. *Consider θ_1 from Figure 2. This SO tgd is trivially consistent and linear, because it consists of only a single clause and the only Skolem function f is just used once. Procedure `SortArgIncl` would create the following order over the quantified variables: $d = p < b$ (b is last because it is not used in any function). The resulting quantifier prefix is $\forall d \forall p \exists v_f \forall b$. Now consider θ_2 with two Skolem terms f and g which will be ordered $f < g$ and, thus, θ_2 is linear. Ordering the variables results in the quantifier prefix $\forall d \exists v_f \forall p \exists v_g \forall b$. Since SO tgds θ_3 and θ_4 are not linear, they can not be rewritten using Algorithm 1.*

Algorithm 1 `Linearize` (θ)

```
Input   :  A SO tgd $\theta$ without Skolem equalities.
Output  :  If successful, a nested GLAV mapping $\Omega$;
           otherwise, $\theta$.
1:  $\mathbf{f} \leftarrow$ the set of Skolem functions in $\theta$
2:  $\Pi \leftarrow$ the maximal partition of $\theta$
    {Check Consistency}
3:  for each $f \in \mathbf{f}$ do
4:      $\pi_f \leftarrow$ block that contains $f$
5:      $arg(f) \leftarrow (t_1, \ldots, t_a)$ for a term $f(t_1, \ldots, t_a)$ in $\pi_f$
6:      for each term $f(t'_1, \ldots, t'_a)$ in $\pi_f$ do
7:          $arg(f)' \leftarrow (t'_1, \ldots, t'_a)$
8:          if $\exists i, j : t'_i = t'_j$ or $arg(f) \neq arg(f)'$ then
9:              return $\theta$ {Not consistent}
    {Check Linearity}
10: for each $\pi$ in $\Pi$ do
11:     $O_\pi \leftarrow$ list of Skolem terms in $\pi$
12:     if not SortArgIncl($O_\pi$) then
13:         return $\theta$ {Not linear}
    {Unskolemize}
14: $\Omega \leftarrow \emptyset$ {nested GLAV}
15: for each $\pi$ in $\Pi$ do
16:     $\Omega \leftarrow \Omega \cup$ ReduceToNested($\pi$,$O_\pi$)
17: return $\Omega$
```

5. LINEARIZATION USING FDS

The `Linearize` algorithm fails if two Skolem terms in a block have unrelated (by set containment) argument sets. However, in many cases there is a hidden connection between these arguments based on constraints that hold on the source schema. In this section, we present the main contribution of this paper, an algorithm for rewriting SO tgds in the presence of source keys or FDs. We use these constraints to augment Skolem terms with additional arguments, thus increasing the chance of rewriting. We first prove that this type of augmentation preserves equivalence. Then we introduce our extended rewriting algorithm and prove its correctness. Moreover, we show that augmentation of Skolem arguments never transforms a linearizable SO tgd (or more precisely one on which our Algorithm `Linearize` would succeed) into an SO tgd that is not linearizable by our algorithm, i.e., using the source constraints can never hurt our ability to find a rewriting.

5.1 Equivalence Preserving Augmentation of Skolem Terms based on Source FDs

Consider a consistent SO tgd θ and a set Σ_S of source FDs over $\mathbf{S}$. Each FD fd is of the form $fd : X \rightarrow Y$ where X, Y and U are vectors of attributes; $X, Y \subseteq U$ and U is the set of all attributes over a given source relation. The intuition of using FDs to augment Skolem terms is that if a Skolem term $f(\mathbf{x})$ contains variables corresponding to the left-hand-side of a source FD fd, then adding variables corresponding to the right-hand-side of fd to the list of arguments of f, is an *equivalence preserving transformation*. Adding additional arguments to Skolem terms can help us rewrite SO tgds into nested GLAV mappings, as we show next.

EXAMPLE 5.1. *Recall θ_3 from Example 1.3 over source relation* `City` *and target relation* `Location`*, with disjoint Skolem terms f and g. This SO tgd is consistent but not linear. Assume that* `Department` $\rightarrow$ `City` *holds over relation* `City`*. If we apply this FD over Skolem term f, then by adding variable c to f we can rewrite θ_3 as an equivalent formula*

$$\exists f \exists g\,(\,\forall d \forall c\; C(d,c) \rightarrow L(d, f(d,c), c, g(c))\,)$$

This equivalent consistent and linear SO tgd can be rewritten into a nested GLAV mapping using Algorithm 1.

Intuitively, the type of augmentation exemplified above can be applied to all occurrences of a Skolem term $f(\mathbf{x})$ only if the base FDs in Σ_S imply an FD $fd : \mathbf{x} \rightarrow \mathbf{y}$ over the variables in *each* clause that references f. When this condition is fulfilled, we can safely replace $f(\mathbf{x})$ with $f(\mathbf{x}, \mathbf{y})$ in all clauses containing f. Let C be a clause of an SO tgd. We use Σ_C, called the *clausal FDs* for C, to denote the set of FDs that are implied by Σ_S over the variables of the left-hand-side of the clause (the source expression). Of course it is well known how to compute Σ_C for FDs over conjunctive expressions [19]. Next, we show that augmentation based on implied source FDs preserves equivalence.

THEOREM 5.2. *Let θ be a consistent SO tgd, Σ_S a set of source FDs, and $f(\mathbf{x})$ a Skolem term used in θ. Moreover, let π_f be the set of clauses of θ that mention f, and let $\mathbf{x}' \rightarrow \mathbf{y}$ with $\mathbf{x}' \subseteq \mathbf{x}$ be an implied clausal FD that holds for all clauses in π_f. Then the following equivalence holds:*

$$\theta[f(\mathbf{x}) \leftarrow f(\mathbf{x}, \mathbf{y})] \cup \Sigma_S \equiv \theta \cup \Sigma_S$$

There may be cases where applying FDs will not help us achieve a successful rewriting of SO tgds. However, as we show in the experimental evaluation, in practice this approach is very effective. This is because the arguments of Skolem functions are usually not chosen randomly, but rather in a principled way to encapsulate certain desired grouping or correlation semantics [1, 22, 23, 29].

5.2 Algorithm and Correctness Proof

We now introduce a polynomial time algorithm for rewriting SO tgds in the presence of source FDs. Algorithm 2, called `LinearizeFDs` (θ, Σ_S), takes as input a consistent SO tgd θ in clausal normal form and a set Σ_S of source FDs. The algorithm consists of two main steps, augmenting Skolem terms based on implied FDs and applying our rewriting algorithm to the result of this equivalent transformation.

As before, we generate the maximal partition of θ (Line 2). Lines 3 and 4 compute the clausal FDs Σ_C (procedure `InferFDs`) that hold over the variables of a clause. Afterwards (Lines 5 to 8), for each Skolem function f, we retrieve the block π_f that contains f and compute Σ_f, the set of FDs that holds over all clauses in π_f. Recall from Theorem 5.2 that a Skolem function can be augmented based on FDs that hold over all clauses using f. The clauses in block π_f are clearly a superset of these clauses. Thus, by intersecting the sets of FDs from the clauses in π_f, we know that each FD in Σ_f can be used to augment the arguments of f. Method `VariableClosure` is used to replace the arguments $arg(f)$ of f with their attribute closure according to Σ_f. This step is equivalent to a repeated application of our equivalence preserving transformation and thus, generates an SO tgd that is equivalent to the input SO tgd. Last, we run the rewriting algorithm over the resulting SO tgd θ^{aug} (Line 9).

Algorithm 2 `LinearizeFDs` (θ, Σ_S)

```
Input  : A consistent SO tgd $\theta$ without Skolem equalities
         A set $\Sigma_S$ of FDs over source schema $\mathbf{S}$
Output : If successful, a nested GLAV mapping $\Omega$;
         otherwise, $\theta$.
1: $\theta^{aug} \leftarrow \theta$
2: $\Pi \leftarrow$ maximal partition of $\theta^{aug}$
   {Compute FDs for each Clause}
3: for each clause $C$ in $\theta^{aug}$ do
4:     $\Sigma_C \leftarrow$ InferFDs$(C, \Sigma_S)$
   {Augment Skolem Terms}
5: for each Skolem $f$ in $\theta$ do
6:     $\pi_f \leftarrow$ block that contains $f$
7:     $\Sigma_f \leftarrow \bigcap_{C \in \pi_f} \Sigma_C$ {FDs valid for all Clauses in $\pi_f$}
8:     $arg(f) \leftarrow$ VariableClosure$(arg(f), \Sigma_f)$
   {Run Linearization Algorithm}
9: return Linearize$(\theta^{aug})$
```

EXAMPLE 5.3. *Consider the SO tgd θ_4 from Figure 2 which joins relations* `WorksOn` *and* `Audit` *on the attribute* `BudgetId`*. The source FDs,* `Department`, `Project` $\rightarrow$ `BudgetId` *and* `BudgetId` $\rightarrow$ `Auditor`*, together with the reuse of variable b (the join) imply the FD $d, p \rightarrow b, a$ over the source expression. This (implied) dependency enables us to augment f with variable a to produce the rewriting*

$$W(d,p,b) \wedge A(b,a) \rightarrow B(p, f(d,p,b,a), g(b,a)),$$

where as before we assume existential quantification over Skolems and universal quantification over source variables. This equivalent augmented SO tgd is now linear.

Proving the correctness of Algorithm 2 amounts to showing that the transformed SO tgd θ^{aug} is equivalent to the input θ, under the assumption that the source FDs Σ_S hold.

THEOREM 5.4. *Let θ be a consistent SO tgd without equalities between or with Skolem terms and let Σ_S be a set of source FDs, then*

$$\texttt{LinearizeFDs}(\theta, \Sigma_S) \cup \Sigma_S \equiv \theta \cup \Sigma_S.$$

According to Theorem 5.4, the rewritten SO tgd θ^{aug} produced by `LinearizeFDs` is equivalent to the input. However, does this algorithm also preserve the sufficient condition?

THEOREM 5.5. *Let θ be a consistent SO tgd, Σ_s a set of source FDs, and θ^{aug} be the equivalent SO tgd derived by algorithm* `LinearizeFDs`*. If θ can be rewritten by* `Linearize` *so can θ^{aug}. The opposite direction does not hold in general.*

5.3 Discussion

The algorithm presented in this section divides the task of rewriting in two steps: (1) apply an equivalence preserving re-Skolemization strategy, and (2) check the linearity condition on the result. In the first step, we could use any re-Skolemization strategy as long as it preserves equivalence. Of course, the first re-Skolemization strategy that comes to mind is that of Marnette et al. [23], which has been applied in the context of efficiently computing compact data exchange solutions under target constraints using SQL queries. This sound but not complete strategy uses source FDs to minimize the arguments of Skolem functions in a GLAV mapping, and uses target FDs to choose the most general Skolemization among competing ones.[2] We can take the first part

[2]Like ours, the intuition behind this technique is that of injectively creating and equating labelled `NULLs`.

Name	Description	Example	Skolems
ADD	Copy a relation and add new attributes	$R(a, b) \rightarrow S(a, b, f(a, b))$	Variable
ADL	Copy a relation, add and delete attributes in tandem	$R(a, b) \rightarrow S(a, f(a))$	Variable
CP	Copy a relation	$R(a, b) \rightarrow S(a, b)$	-
DL	Copy a relation and delete attributes	$R(a, b) \rightarrow S(a)$	-
HP	Horizontally partition a relation into multiple relations	$R(a, b) \wedge a = c_1 \rightarrow S_1(b)$ $R(a, b) \wedge a = c_2 \rightarrow S_2(b)$	-
ME	Inverse of vertical partitioning (merge)	$R(a, b) \wedge S(b, c) \rightarrow T(a, b, c)$	-
MA	Inverse of vertical partitioning + adding attributes	$R(a, b) \wedge S(b, c) \rightarrow T(a, b, c, f(a, b, c))$	Variable
OF	Object fusion, e.g., inverse of horizontal partitioning	$R(\underline{a}, b) \rightarrow T(a, b, f(a))$ $S(\underline{a}, c) \rightarrow T(a, g(a), c)$ $R(\underline{a}, b) \wedge S(\underline{a}, c) \rightarrow T(a, b, c)$	Fixed
SJ	Copy relation (S) and create a relationship table (T) through a self-join	$R(\underline{a}, b, c) \rightarrow S(a, c)$ $R(\underline{a}, b, c) \wedge R(\underline{b}, d, e) \rightarrow T(a, b)$	-
SU	Copy a relation and create a surrogate key	$R(a, b) \rightarrow S(\underline{f(a, b)}, b, g(b))$	Fixed / Variable
VH	Vertical partitioning into a HAS-A relationship	$R(a, b) \rightarrow S(\underline{f(a)}, a) \wedge T(\underline{g(a, b)}, b, f(a))$	Fixed
VI	Vertical partitioning into an IS-A relationship	$R(\underline{a}, b, c) \rightarrow S(\underline{a}, b) \wedge T(\underline{a}, c)$	-
VNM	Vertical partitioning into an N-to-M relationship	$R(a, b) \rightarrow S_1(\underline{f(a)}, a) \wedge M(\underline{f(a)}, \underline{g(b)}) \wedge S_2(\underline{g(b)}, b)$	Fixed
VP	Vertical partitioning	$R(a, b) \rightarrow S_1(\underline{f(a, b)}, a) \wedge S_2(\underline{f(a, b)}, b)$	Variable

Figure 3: Mapping Primitives

of this strategy and use it to replace the argument set $arg(f)$ of a Skolem function f with a minimal subset $min(args(f))$ such that both share the same closure according to the implied source FDs. We call this new strategy `LinearizeMin`. We ask, then, what are the implications of minimizing instead of maximizing the argument sets?

Example 5.6. *Recall again Ex. 4 from Figure 2 with Skolem functions $f(d, p, b)$ and $g(b, a)$. If we minimize these, we obtain $f(d, p)$ and $g(b)$, respectively. The resulting formula is not linear, in contrast to the re-Skolemized formula we would obtain using `LinearizeFDs`. Now, consider a linear variation of this same example where $f(d, p, b)$ is replaced by $f(d, p, b, a)$ and $g(b, a)$ stays the same. Applying minimization, the re-Skolemizations for Skolems f and g are $f(d, p)$ and $g(b)$ which again are not linear.*

This example demonstrates that our approach of maximizing argument sets can solve cases for which minimization fails and, even more important, the example also shows that minimization may turn a linear SO tgd into a non-linear one. The following theorem proves that our approach is strictly better than minimization for the purpose of translating SO tgds into (nested) GLAV mappings.[3]

Theorem 5.7. *Let θ be a consistent SO tgd and Σ_s a set of source FDs. Let θ^{Min} be the result of minimizing the argument sets of each Skolem f in θ. Furthermore, let θ^{aug} be the equivalent SO tgd derived by algorithm `LinearizeFDs`. If θ^{min} can be rewritten by `Linearize` so can θ^{aug}.*

6. EXPERIMENTS

To evaluate our techniques, we used STBenchmark 2.0 [7], a benchmark for generating SO tgds that extends the well-known STBenchmark [2], a GLAV mapping benchmark. The main motivation of our evaluation is to put into practice our rewriting algorithms for SO tgds, and to compare against the state-of-the-art using an implementation of Nash et al.'s [26] sufficient condition. This sufficient condition, which we call `NBM` after the authors, requires consistency and that every universally quantified source variable that appears in the target expression (right-hand-side expression) of an SO tgd also appears as an argument in every consistent Skolem function. We also compare against the minimization strategy [23] discussed in Section 5.3 (`LinearizeMin`), which minimizes the argument sets of Skolem functions using implied FDs before checking our linearity condition. Using STBenchmark 2.0, we generate a large number of SO tgds, and we measure the success of rewriting into nested GLAV (referred to as *success rate*) independently for each technique, i.e., `NBM`, `Linearize`, `LinearizeFDs`, and `LinearizeMin`. We start by describing the features of STBenchmark 2.0 that we use, and then describe our experimental study.

6.1 Mapping Generator

STBenchmark 2.0 [7] is able to randomly generate a broad range of SO tgds by combining basic mapping primitives into complex mappings. The mapping generator takes as input a set of configuration parameters and returns as output a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$. The benchmark supports different variants of vertical partitioning primitives, some of which offer specialized support for generating mappings with (possibly repeated) Skolem functions, including functions with both overlapping and disjoint arguments. This permits the modeling of complex correlations, including Skolem functions that cannot be unskolemized and rewritten into GLAV. In Figure 3, we outline the benchmark mapping primitives used in our study. For each primitive, we offer a brief description and an example mapping, in which we underline those attributes which are part of primary keys, when these are essential for understanding the semantics of the transformation. Moreover, we indicate whether each primitive generates no Skolem terms ($-$), or some Skolem terms, using fixed or variable Skolemization strategies (*Fixed* and *Variable*, respectively). Note that the original STBenchmark [2] only uses Skolem functions in surrogate key (SU) and vertical partitioning (VP) scenarios, and only a single function in each. Hence, the original STBenchmark scenarios could always be unskolemized (i.e., they were nested GLAV), whereas STBenchmark 2.0 generates SO tgds. Note that the example mappings in the figure represent the simplest version of each primitive. The generated mappings may be much more involved and diverse. For instance, a vertical partitioning may split a relation into more than two

[3]This is not surprising because the minimization strategy was not designed with unskolemization in mind.

Parameter	Min	Max
Number of Primitives (per type)	0	10
Number of Relations Per Schema	1	100+
Number of Attributes Per Relation	2	15
Number of Key Attributes	1	3
Join Path Length	2	4
Join Type	Star	Chain
Primary Key FDs	No	Yes
Source FDs	0%	50%
Skolem Noise	0%	50%
Source Reuse	0%	50%

Figure 4: Some Configuration Parameters

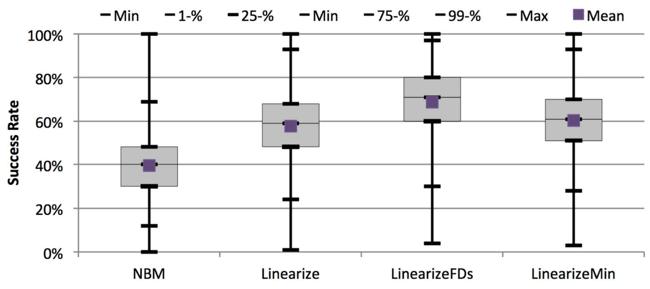

Figure 5: 12,500,000 Random Schema Mappings

fragments (depending on how the configuration parameters are set). For primitives that support variable Skolemizations, we use three different strategies (i.e., *Key*, *All*, and *Random* chosen through a parameter called *Skolem Mode*).

We made use of STBenchmark 2.0's ability to support the generation of primary keys, and random multi-attribute and partial FDs over the source. Configuration parameters, *Primary Key FDs* and *Source FDs*, are used to turn these features on and off as desired. We also made extensive use of the benchmark's ability to reuse schema elements across primitives. For example, two instances of a CP primitive may copy from the same source relation. This is of importance for generating correlations among mappings and in a sense, more realistic cases of Skolem terms across clauses used in the complex SO tgds we wish to consider.

Composition of schema mappings, like schema evolution, can lead to complex interactions between Skolem terms (specifically the arguments of the Skolem functions). In addition to Skolem terms introduced in the benchmark's primitives, we introduce additional Skolem terms by randomly choosing source attributes to be replaced (in the target) with Skolem functions (using the benchmark's configuration parameter *Skolem Noise* [7]). For instance, assume the Skolem term $f(a)$ was assigned to attribute b of relation $S(a, b)$. We would transform the SO tgd $S(x_1, x_2) \rightarrow T(x_1, x_2)$ into $S(x_1, x_2) \rightarrow T(x_1, f(x_1))$.

6.2 Random Scenarios

To analyze the effectiveness of the four rewriting techniques (NBM, Linearize, LinearizeFDs and LinearizeMin), we randomly generated a broad and large number of schema mappings using STBenchmark 2.0 [7]. Figure 4 outlines some of the most important configuration parameters involved in this experimental run and their valid ranges. For example, we randomly selected the number of instances for each mapping primitive (from 0 to 10, following a uniform distribution). The size of the source and target schema was determined not only by the number of requested attributes per relation (plus/minus deviations), but also by the type of requested primitives and some other additional parameters, such as the length of join paths. We consider variable types of joins (i.e., star and chain), key sizes, percentages of source FDs, Skolem noise, and source reuse.

To understand the stability of our experiments over these random configurations, we initially repeated each experiment up to 1,000 times. For all four techniques, we found that the values were already stable at 100 repetitions per configuration (less than 1% difference for each percentile of the distribution, with most percentiles being identical). For all experiments, the reported data values are averaged across 250 runs. This guarantees stable results for the whole breadth of schema mappings that can be generated with STBenchmark 2.0 (using these parameter ranges). Figure 5 plots the distribution of the success rate (i.e., minimum, 1-percentile, 25-percentile, median, 75-percentile, 99-percentile, maximum) for the four techniques, over a total of 12,500,000 schema mappings. By *success rate*, we mean the fraction of SO tgds from a schema mapping that were successfully rewritten, e.g., for a schema mapping with 10 SO tgds, 30% would indicate that 3 SO tgds were successfully rewritten. For a few mappings, all methods performed very well or very bad. Given the random nature of the generated schema mappings, this was an expected outcome. Recall that even the types of primitives used in the mappings were chosen randomly. Figure 5 shows that `NBM` is already able to rewrite 40% of the scenarios, `Linearize` can rewrite almost 50% more achieving a 58% success rate while `LinearizeFDs` achieves an average 71% success rate. The success rate of `LinearizeMin` is only slightly higher than `Linearize`. We will thus only consider the latter technique over real-life scenarios on Section 6.6. These results suggest that significant gains are possible by exploiting our more general rewriting techniques and the presence of source FDs.

6.3 Mapping Primitives

Next we ran experiments using only one primitive at a time to understand how the techniques deal with different types of real world scenarios. Figures 6 and 7 present the average success rate (over 250 runs) for each primitive with 0% and 50% Skolem noise, respectively. In both cases, we use 50% of source FDs. In the case of 0% Skolem noise, we found that all methods perform very well, i.e., we achieve 100% success rate for many primitives. This can be attributed to the fact that 6 of the 14 mapping primitives do not generate any Skolem terms (see Figure 3, e.g., CP and DEL). `Linearize` and `LinearizeFDs` perform better for the "harder" primitives. In particular, `LinearizeFDs` is the only technique that can successfully deal with vertical partitioning into N-to-M relationships (VPN). This success is due to the presence of source FDs which can be exploited to linearize Skolem terms with disjoint argument sets. As revealed by Figure 7, the importance of applying FDs is very noticeable in the case of 50% Skolem noise (i.e., high prevalence of Skolem functions with random arguments). For many primitives, `LinearizeFDs` succeeds in over 80% of the cases, `Linearize` in about 40%, and `NBM` in about 20%. However, some primitives result in Skolem terms that are more difficult to rewrite than others. Notably, merge primitives (i.e., MA and ME, which encode denormalization or the reversal of vertical partitioning) may use long sequences of joins in

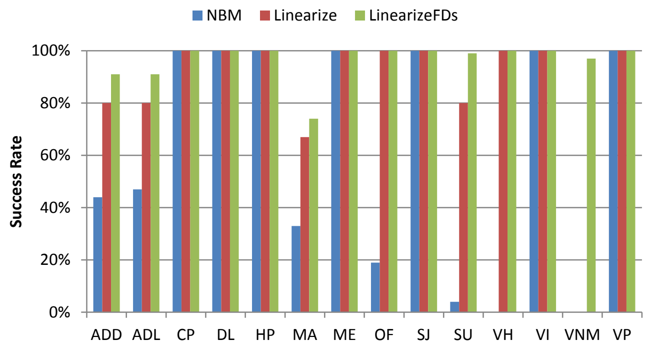

Figure 6: Per Primitive with 0% Skolem Noise

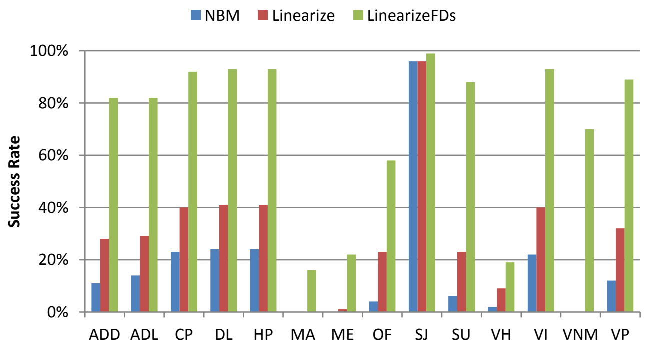

Figure 7: Per Primitive with 50% Skolem Noise

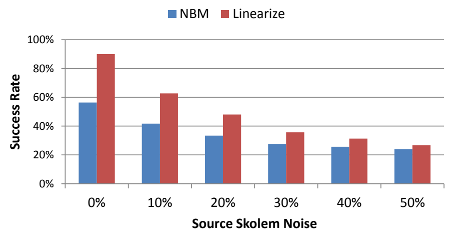

Figure 8: Increasing Skolem Noise

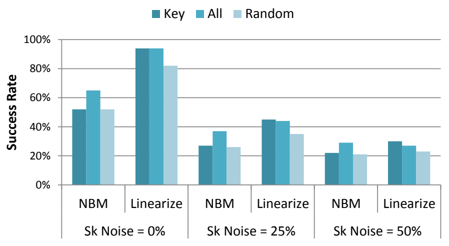

Figure 9: Skolem Mode vs. Noise

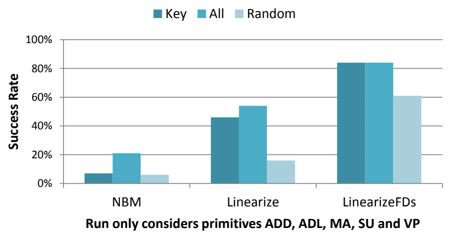

Figure 10: Varying Skolem Modes

a single SO tgd. The longer the join sequence, the higher the chance that randomly introduced Skolem terms are not rewritable, not even using FDs. We also found that, in general, `NBM` achieves lower success rates for higher percentages of Skolem noise and, in particular, for most vertical partitioning primitives. This finding aligns with the results reported by Bernstein et al. [8] who also point out the difficulty of rewriting these types of primitives.[4]

6.4 Amount and Type of Skolem Terms

We now investigate how the amount and types of Skolem terms influence success rate. Recall that in STBenchmark 2.0 [7], the configuration parameter *Skolem Noise* simulates that the mapping scenario is the result of composition or correlation operations. The SO tgd mapping generator supports three strategies for choosing the arguments of Skolem functions. *Key* uses the key attributes of a source expression as arguments for a given Skolem function; *All* uses all attributes of a source expression; and *Random* randomly selects a number of attributes. Figure 8 shows the average success rates over all Skolem modes for techniques `NBM` and `Linearize`, with Skolem noise varying from 0% to 50%. FDs were deactivated for this experiment. In general, we observe that the performance of both techniques decreases as we increase the amount of Skolem noise. This was expected, because intuitively the more Skolem noise is present, the higher the chance that an SO tgd contains Skolems that can not be rewritten. `Linearize` succeeds in over 80% percent of the cases for 0% Skolem noise and only in 27% of the cases for 50% Skolem noise. Note that 50% of Skolem noise causes an unrealistically large number of Skolem terms that are unlikely to appear in practical schema mappings. Nonetheless, we included these measurements to understand how the techniques behave under extreme conditions.

[4]Note that their experiments measure the success of composition rather than the success rate of rewriting [8].

Figure 9 shows the breakdown for each of the three Skolem modes (i.e., Key, All and Random). As can be seen, `NBM` achieves the best success rate if all attributes of a relation (*All*) are used as Skolem arguments, and exhibits similar behavior for *Random* and *Key* Skolem arguments. This behavior stems from the fact that *NBM*'s sufficient condition explicitly requires all exchanged source variables to appear as arguments to every Skolem term, a condition that is easily met under mode *All*. In contrast, `Linearize` works similarly well for both modes *Key* and *All*.

Not all primitives generate Skolem terms and for some primitives the Skolem arguments are predetermined by the semantics of the mapping transformation (i.e., Fixed vs. Variable Skolemization, as outlined in the last column of Figure 3). In the next experiment, we consider only those primitives that support variable Skolemization strategies (i.e., AD, ADL, MA, SU, and VP). Using a fixed Skolem noise percentage of 25%, we compare the success rate of the three techniques for the different Skolem strategies. We show the results of this experiment in Figure 10. Notice that `NBM` performs poorly, in particular for random Skolem terms (about 3% success rate). The success rates for `Linearize` and `LinearizeFDs` are better in general, but are also significantly lower for *Random* (about 20%).

6.5 Impact of Primary Keys and FDs

As part of our experimental plan, we also studied the effect of key and functional dependencies in rewriting SO tgds. We varied three main configuration parameters: *Skolem Mode* (i.e., Key, All and Random), Percentage of *Source FDs* (i.e., 0%, 25%, and 50%), and *Primary Key FDs*. Each separate run considered all mapping primitives, variable type of joins, and primary keys between 1 and 3 attributes. Source Skolem noise was fixed to 50%. We generated mappings with and without primary key FDs, and also with various degrees of additional source FDs. The results are depicted in Figure 11. We include `Linearize` as a baseline (the best we can do

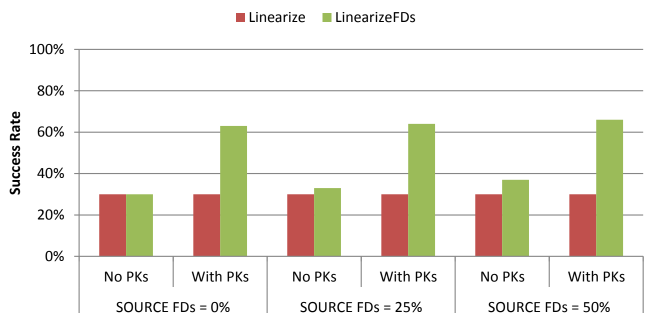

Figure 11: Impact of Primary Keys

Amalgam

Scenario	NBM	Linearize	LinearizeFDs	LinearizeMin
$S_1 \to S_2$	0	3	8	3
$S_1 \to S_3$	1	8	8	8
Total (16 SO tgds)	**6%**	**68%**	**100%**	**68%**

Bio

Scenario	NBM	Linearize	LinearizeFDs	LinearizeMin
GUS $\to$ BioSQL	0	4	7	4
Total (8 SO tgds)	**0%**	**50%**	**87%**	**50%**

3SDB

Scenario	NBM	Linearize	LinearizeFDs	LinearizeMin
$S_1 \to S_2$	4	6	10	9
$S_2 \to S_3$	7	9	9	9
$S_1 \to S_3$	2	4	11	4
Total (30 SO tgds)	**43%**	**63%**	**100%**	**73%**

Figure 12: Real-world Mappings

without using FDs). As we increase the percentage of source FDs, there is a slight increase (5% for 25% FDs and 10% for 50% FDs) in the success rate of `LinearizeFDs`. This experiment shows that primary keys act as a catalyst for rewriting, yielding at least a 75% increased success rate. This finding confirms our hypothesis about the advantages of considering FDs for rewriting.

6.6 Real-World Mappings

Last, we applied our techniques over three real-life mapping scenarios. The first scenario, *Amalgam* [25], consists of schemas describing scientific bibliographies. The second and third scenarios, *Bio* [3] and *3SDB* [4], are from the biological domain. We created SO mappings by relying on the semantics of the schemas and documented data transformations. We compared all rewriting techniques including `LinearizeMin`. Figure 12 reports the number of SO tgds that were successfully rewritten by each technique.

Amalgam. This scenario comprises two mappings between schemas S_1 and S_2, and between S_1 and S_3. Schemas S_1, S_2, and S_3 consists of 15, 27, and 5 relations, respectively. Each mapping is specified by 8 SO tgds, and all Skolem functions include keys. `LinearizeFDs` succeeds on all SO tgds, followed by `Linearize` and `LinearizeMin` with a success rate of 68%. `NBM` only succeeds for one SO tgd as the remaining ones require a more expressive mapping language.

Bio. We use a mapping between fragments of the Genomics Unified Schema *GUS* and the generic relational Biological Schema *BioSQL*.[5] The source schema *GUS* consists of 7 relations, and the target schema *BioSQL* consists of 6 relations. The schema mapping is specified by 8 SO tgds. `LinearizeFDs` succeeds on rewriting all but one SO tgd due to the presence of disjoint Skolem argument sets, thus outperforming the other techniques by at least 74%. `NBM` did not succeed on any of the tgds.

3SDB. This scenario consists of three mappings representing the evolution of three versions (S_1, S_2, and S_3) of a biological sample database schema used for gene expression analysis [4]. All rewriting techniques perform reasonably well, in particular with respect to the second mapping which consists mostly of full tgds with fewer Skolem functions. The third mapping from S_1 to S_3 models the composition of the evolution mappings and exhibits complex Skolem argument sets, resulting in a lower success rate for all techniques except `LinearizeFDs`. Notice that while `LinearizeMin` accomplishes similar performance as `LinearizeFDs` for the simpler evolution mappings, it only succeeds in rewriting up to 36% of the tgds in the composed mapping.

[5]From www.gusdb.org and www.biosql.org, respectively.

6.7 Discussion

In summary, our technique `LinearizeFDs` consistently outperforms both `Linearize` and `NBM` by a large margin over both synthetic and real-world mappings (e.g., over 95% success rate in the latter). The performance of `LinearizeFDs` is largely improved by the existence of primary keys in the source. Using implied FDs to *augment* Skolem terms appears to be more beneficial than using FDs to minimize Skolem terms [23] before checking our linearity condition (i.e., `LinearizeMin`). Since primary keys are available in most schemas or can be added without much effort, we expect our method to perform very well in practice.

7. RELATED WORK

The *NBM* algorithm [26] was used as part of a mapping composition framework presented by Bernstein et al. [8]. This work presented a composition algorithm based on view unfolding and a comprehensive experimental study in the context of schema evolution (whereas our work and evaluation considers general SO tgds). In Bernstein et al., source key dependencies were used to minimize the argument sets of Skolem functions (for example, a Skolem function $f(x, y)$ could be minimized to $f(x)$ if $x \to y$). We have found that such minimization can sometimes hide a linearization opportunity and instead use an algorithm that augments function arguments using FDs. Importantly, Bernstein et al. [8] suggest as future work that it may be possible to exploit key dependencies "in a more direct way". We have confirmed their conjecture and also shown how to leverage more general source FDs along with dependencies implied by a mapping.

A number of techniques, logical and otherwise, have been proposed for reducing the argument sets of Skolem functions in SO tgds. Using SO Skolemization [10], Fagin et al. [12] show that every s-t tgd is equivalent to an SO tgd without equalities. It is often possible to apply a prior step to this transformation that leads to Skolem terms with fewer arguments. Practically, this means obtaining SO tgds with Skolem terms depending *only* on the universal variables that are effectively being exchanged, instead of using all universal variables as dictated by the standard Skolemization strategy [10]. Yu and Popa [32] exploit this optimization, among others, in their mapping composition algorithm and use it to simplify SO tgds by replacing every equality between two Skolem terms with an equality of their arguments. This simplification guarantees chase equivalence. In the context of exchanging data in open and closed worlds, Libkin and Sirangelo [22] discuss the use of *custom* Skolemization strategies to encapsulate the semantics of 1-to-1 and 1-to-n relationships over target attributes. Our approach of using FDs to modify the arguments sets of Skolem functions may be seen as proposing yet another Skolemization strategy, albeit one that has the property of generating more opportunities for the successful unskolemization of the resulting SO tgd.

8. CONCLUSIONS

We introduced two approaches for transforming SO tgds into equivalent nested GLAV mappings. Our approach assumes we are given an (arbitrary) SO tgd (without Skolem equalities) and provides sufficient conditions for when the rich Skolem functions in SO tgds are well-behaved and have an FO semantics. We experimentally showed that these conditions are able to handle a very large number of real schema mappings. Going forward, we would like to incorporate our insights on both linearization and using FDs directly into mapping operators including composition and MapMerge. This could lead to more concise Skolemizations and in general a better understanding of how to determine the arguments of newly introduced Skolem functions within mappings. This follows a trend of using FDs to simplify data exchange [23], but holds the promise of being able to use FDs and more general Skolem management techniques directly within a host of model management operators.

Acknowledgments. We thank the anonymous referees for their valuable suggestions. Special thanks to Sheila McIlraith for relevant discussions, and to Bodgan Alexe and Yuan An for providing us with real-life schemas and mappings. Arocena was supported by the NSERC Business Intelligence Network.

9. REFERENCES

[1] B. Alexe, M. A. Hernández, L. Popa, and W. C. Tan. MapMerge: Correlating Independent Schema Mappings. *VLDB J.*, 21(2):191–211, 2012.

[2] B. Alexe, W. C. Tan, and Y. Velegrakis. STBenchmark: Towards a Benchmark for Mapping Systems. *PVLDB*, 1(1):230–244, 2008.

[3] B. Alexe, B. ten Cate, P. Kolaitis, and W. Tan. Designing and refining schema mappings via data examples. In *SIGMOD*, pages 133–144, 2011.

[4] Y. An, A. Borgida, R. J. Miller, and J. Mylopoulos. A Semantic Approach to Discovering Schema Mapping Expressions. In *ICDE*, pages 206–215, 2007.

[5] M. Arenas, R. Fagin, and A. Nash. Composition with Target Constraints. *Logical Methods in Comput. Sci.*, 7(3), 2011.

[6] M. Arenas, J. Pérez, J. Reutter, and C. Riveros. Inverting Schema Mappings: Bridging the Gap between Theory and Practice. *PVLDB*, 2(1):1018–1029, 2009.

[7] P. C. Arocena, M. D'Angelo, B. Glavic, and R. J. Miller. STBenchmark 2.0. Technical report, University of Toronto, 2013. `http://dblab.cs.toronto.edu/project/STBench2.0`.

[8] P. A. Bernstein, T. J. Green, S. Melnik, and A. Nash. Implementing Mapping Composition. *VLDB J.*, 17(2):333–353, 2008.

[9] P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger. A Vision for Management of Complex Models. *SIGMOD Record*, 29(4):55–63, Dec. 2000.

[10] H. B. Enderton. *A Mathematical Introduction to Logic.* Harcout Academic Press, 2nd. Edition, 2001.

[11] R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[12] R. Fagin, P. Kolaitis, L. Popa, and W. C. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. *TODS*, 30(4):994–1055, 2005.

[13] I. Feinerer, R. Pichler, E. Sallinger, and V. Savenkov. On the Undecidability of the Equivalence of Second-Order Tuple Generating Dependencies. In *AMW*, 2011.

[14] A. Fuxman, M. A. Hernández, H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested Mappings: Schema Mapping Reloaded. In *VLDB*, pages 67–78, 2006.

[15] A. Fuxman, P. Kolaitis, R. J. Miller, and W. C. Tan. Peer Data Exchange. *TODS*, 31(4):1454–1498, 2006.

[16] D. Gabbay, R. Schmidt, and A. Szalas. *Second Order Quantifier Elimination: Foundations, Computational Aspects and Applications.* College Publications, 2008.

[17] T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. ORCHESTRA: Facilitating Collaborative Data Sharing. In *SIGMOD*, pages 1131–1133, 2007.

[18] R. Hull and M. Yoshikawa. ILOG: Declarative Creation and Manipulation of Object Identifiers. In *VLDB*, pages 455–468, 1990.

[19] A. Klug and R. Price. Determining View Dependencies Using Tableaux. *TODS*, 7(3):361–380, 1982.

[20] M. K. Lawrence, R. A. Pottinger, and S. Staub-French. Data Coordination: Supporting Contingent Updates. *PVLDB*, 4(11):831–842, 2011.

[21] M. Lenzerini. Data Integration: a Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[22] L. Libkin and C. Sirangelo. Data Exchange and Schema Mappings in Open and Closed Worlds. *J. Comput. Syst. Sci.*, 77(3):542–571, 2011.

[23] B. Marnette, G. Mecca, and P. Papotti. Scalable Data Exchange with Functional Dependencies. *PVLDB*, 3(1):105–116, 2010.

[24] B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro. ++Spicy: an OpenSource Tool for Second-Generation Schema Mapping and Data Exchange. *PVLDB*, 4(12):1438–1441, 2011.

[25] R. J. Miller, D. Fisla, M. Huang, D. Kymlicka, F. Ku, and V. Lee. The Amalgam Schema and Data Integration Test Suite. `www.cs.toronto.edu/~miller/amalgam`, 2001.

[26] A. Nash, P. A. Bernstein, and S. Melnik. Composition of Mappings Given by Embedded Dependencies. *TODS*, 32(1):4, 2007.

[27] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In *VLDB*, pages 413–424, 1996.

[28] R. Pichler and S. Skritek. The Complexity of Evaluating Tuple Generating Dependencies. In *ICDT*, pages 244–255, 2011.

[29] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.

[30] L. Seligman, P. Mork, A. Y. Halevy, K. P. Smith, M. J. Carey, K. Chen, C. Wolf, J. Madhavan, A. Kannan, and D. Burdick. OpenII: an Open Source Information Integration Toolkit. In *SIGMOD*, pages 1057–1060, 2010.

[31] B. ten Cate and P. Kolaitis. Structural Characterizations of Schema-Mapping Languages. In *ICDT*, pages 63–72, 2009.

[32] C. Yu and L. Popa. Semantic Adaptation of Schema Mappings when Schemas Evolve. *VDLB*, pages 1006–1017, 2005.