Relational to RDF Data Exchange in Presence of a Shape Expression Schema

Iovka Boneva, Jose Lozano, Sławek Staworko

CRIStAL–UMR 9189, University of Lille and Inria, F-59000 Lille, France

Abstract. We study the relational to RDF data exchange problem, where the target constraints are specified using Shape Expression schema (ShEx). We investigate two fundamental problems: 1) *consistency* which is checking for a given data exchange setting whether there always exists a solution for any source instance, and 2) constructing a *universal solution* which is a solution that represents the space of all solutions. We propose to use *typed IRI constructors* in source-to-target tuple generating dependencies to create the IRIs of the RDF graph from the values in the relational instance, and we translate ShEx into a set of target dependencies. We also identify data exchange settings that are *key covered*, a property that is decidable and guarantees consistency. Furthermore, we show that this property is a sufficient and necessary condition for the existence of universal solutions for a practical subclass of *weakly recursive* ShEx.

1 Introduction

Data exchange can be seen as a process of transforming an instance of one schema, called the *source schema*, to an instance of another schema, called the *target schema*, according to a set of rules, called *source-to-target tuple generating dependencies* (st-tgds). But more generally, for a given source schema, any instance of the target schema that satisfies the dependencies is a *solution* to the data exchange problem. Naturally, there might be no solution, and then we say that the setting is *inconsistent*. Conversely, there might be a possibly infinite number of solutions, and a considerable amount of work has been focused on finding a *universal solution*, which is an instance (potentially with incomplete information) that represents the entire space of solutions. Another fundamental and well-studied problem is checking *consistency* of a data exchange setting i.e., given the source and target schemas and the st-tgds, does a solution exist for any source instance. For relational databases the consistency problem is in general known to be undecidable [6, 13] but a number of decidable and even tractable cases has been identified, for instance when a set of weakly acyclic dependencies is used [10].

Resource Description Framework (RDF) [2] is a well-established format for publishing linked data on the Web, where *triples* of the form $(subject, predicate, object)$ allow to represent an edge-labeled graph. While originally RDF was introduced schema-free to promote its adoption and wide-spread use, the use of RDF for storing and exchanging data among web applications has prompted the development of schema languages for RDF [3, 17, 19]. One such schema language, under continuous development, is Shape Expressions Schemas (ShEx) [8, 20], which allows to define structural constraints on nodes and their immediate neighborhoods in a declarative fashion.

In the present paper, we study the problem of data exchange where the source is a relational database and the target is an RDF graph constrained with a ShEx schema.

Although an RDF graph can be seen as a relational database with a single ternary relation $Triple$, RDF graphs require using *Internationalized Resource Identifiers* (IRIs) as global identifiers for entities. Consequently, the framework for data exchange for relational databases cannot be directly applied *as is* and we adapt it with the help of *IRI constructors*, functions that assign IRIs to identifiers from a relational database instance. Their precise implementation is out of the scope of this paper and belongs to the vast domain of entity matching [14].

Example 1. Consider the relational database of bug reports in Figure 1, where the relation *Bug* stores a list of bugs with their description and ID of the user who reported the bug, the name of each user is stored in the relation *User* and her email in the relation *Email*. Additionally, the relation *Rel* identifies related bug reports for any bug report.

Bug	*bid*	*descr*	*uid*
	1	Boom!	1
	2	Kaboom!	2
	3	Kabang!	1
	4	Bang!	3

User	*uid*	*name*
	1	Jose
	2	Edith
	3	Steve89

Email	*uid*	*email*
	1	j@ex.com
	2	e@o.fr

Rel	*bid*	*rid*
	1	3
	1	4
	2	4

Fig. 1: Relational database (source)

Now, suppose that we wish to share the above data with a partner that has an already existing infrastructure for consuming bug reports in the form of RDF whose structure is described with the following ShEx schema (where : is some default prefix):

$$\mathsf{TBug} \to \{\text{:descr} :: Lit^{1}, \text{:rep} :: \mathsf{TUser}^{1}, \text{:related} :: \mathsf{TBug}^{*}\}$$
$$\mathsf{TUser} \to \{\text{:name} :: Lit^{1}, \text{:email} :: Lit^{1}, \text{:phone} :: Lit^{?}\}$$

The above schema defines two types of (non-literal) nodes: TBug for describing bugs and TUser for describing users. Every bug has a description, a user who reported it, and a number of related bugs. Every user has a name, an email, and an optional phone number. The reserved symbol Lit indicates that the corresponding value is a literal.

The mapping of the contents of the relational database to RDF is defined with the following logical rules (the free variables are implicitly universally quantified).

$$\begin{aligned}
\textit{Bug}(b,d,u) &\Rightarrow \textit{Triple}(\textit{bug2iri}(b), \text{:descr}, d) \wedge \mathsf{TBug}(\textit{bug2iri}(b)) \wedge \\
&\quad\ \textit{Triple}(\textit{bug2iri}(b), \text{:rep}, \textit{pers2iri}(u)) \\
\textit{Rel}(b_1,b_2) &\Rightarrow \textit{Triple}(\textit{bug2iri}(b_1), \text{:related}, \textit{bug2iri}(b_2)) \\
\textit{User}(u,n) &\Rightarrow \textit{Triple}(\textit{pers2iri}(u), \text{:name}, n) \wedge \mathsf{TUser}(\textit{pers2iri}(u)) \\
\textit{User}(u,n) \wedge \textit{Email}(u,e) &\Rightarrow \textit{Triple}(\textit{pers2iri}(u), \text{:email}, e) \wedge \textit{Lit}(e)
\end{aligned}$$

On the left-hand-side of each rule we employ queries over the source relational database, while on the right-hand-side we make corresponding assertions about the triples in the target RDF graph and the types of the nodes connected by the triples. The atomic values used in relational tables need to be carefully converted to IRIs with the help of IRI constructors *pers2iri* and *bug2iri*. The constructors can be *typed* i.e., the IRI they introduce are assigned a unique type in the same st-tgd.

We point out that in general, IRI constructors may use external data sources to properly assign to the identifiers from the relational database unique IRIs that identify the object in the RDF domain. For instance, the user Jose is our employee and is assigned the corresponding IRI emp:jose, the user Edith is not an employee but a registered user of our bug reporting tool and consequently is assigned the IRI user:edith, and finally, the user Steve89 is an anonymous user and is assigned a special IRI indicating it anon:3.

Figure 2 presents an RDF instance that is a solution to the problem at hand. We point out that the instance uses a (labeled) null literal $\perp_1$ for the email of Steve89 that is required by the ShEx schema but is missing in our database. □

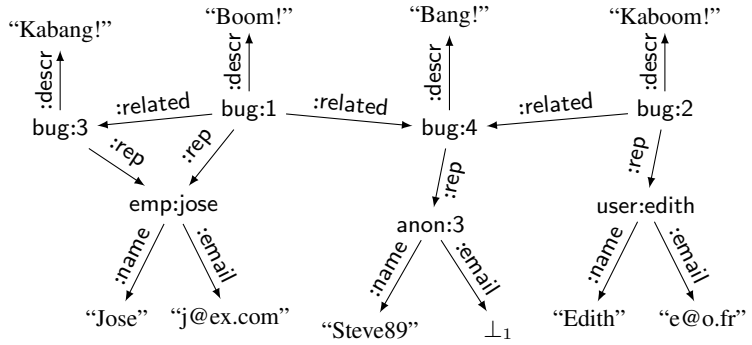

Fig. 2: Target RDF graph (solution)

The presence of target schema raises the question of consistency. On the one hand, we can prove that for any instance of the relational database in Example 1 there exists a target solution that satisfies the schema and the set of source-to-target tuple generating dependencies. On the other hand, suppose we allow a user to have multiple email addresses, by changing the *key* of *Email* to both uid and $email$). Then, the setting would not be consistent as one could construct an instance of the relational database, with multiple email addresses for a single user, for which there would be no solution.

Our investigation provides a preliminary analysis of the consistency problem for relational to RDF data exchange with target ShEx schema. Our contribution can be summarized as follows:

- a formalization of relational to RDF data exchange with target ShEx schema and typed IRI constructors.
- a decidable characterization of a *fully typed key-covered* data exchange setting that is a sufficient and necessary condition for consistency.
- an additional restriction of weak-recursion on ShEx schemas that ensures the existence of universal solution.

Related Work. *Relational Data Exchange, Consistency.* The theoretical foundations of data exchange for relational databases are laid in [4, 10]. Source-to-target dependencies with Skolem functions were introduced by nested dependencies [11] in order to improve the quality of the data exchange solution. General existentially quantified functions are

possible in second order tgds [5]. Consistency in the case of relational data exchange is undecidable, and decidable classes usually rely on chase termination ensured by restrictions such as acyclicity, or guarded dependencies, or restrictions on the structure of source instances. The consistency criterion that we identify in this paper is orthogonal and is particular to the kind of target constraints imposed by ShEx schemas. In [15], static analysis is used to test whether a target dependency is implied by a data exchange setting, these however rely on chase termination. Consistency is an important problem in XML data exchange [4] but the techniques developed for XML do not apply here.

Value Invention, Relational to RDF Data Exchange. Value invention is used in the purely relational setting for generating null values. Tools such as Clio [9] and ++Spicy [16] implement Skolem functions as concatenation of their arguments. IRI value invention is considered by R2RML [1], a W3C standard for writing customizable relational to RDF mappings. The principle is similar to what we propose here. A R2RML mapping allows to specify logical tables (i.e. very similar to left-hand-sides of source-to-target dependencies), and then how each row of a logical table is used to produce one or several triples of the resulting RDF graph. Generating IRI values in the resulting graph is done using templates that specify how a fixed IRI part is to be concatenated with the values of some of the columns of the logical table. R2RML does not allow to specify structural constraints on the resulting graph, therefore the problem of consistency is irrelevant there. In [18], a direct mapping that is a default automatic way for translating a relational database to RDF is presented. The main difference with our proposal and with R2RML is that the structure of the resulting RDF graph is not customizable. In [7] we studied relational to graph data exchange in which the target instance is an edge labelled graph and source-to-target and target dependencies are conjunctions of nested regular expressions. Such a framework raises a different kind of issues, among which is the materialization of a solution, as a universal solution is not necessarily a graph itself, but a graph pattern in which some edges carry regular expressions. On the other hand, IRI value invention is not relevant in such framework.

Organization. In Section 2 we present basic notions. In Section 3 we show how ShEx schemas can be encoded using target dependencies. In Section 4 we formalize relational to RDF data exchange. In Section 5 we study the problem of consistency. And finally, in Section 6 we investigate the existence of universal solutions. Conclusions and directions of future work are in Section 7. The missing proofs can be found in the full version [**?**].

2 Preliminaries

First-order logic. A *relational signature* $\mathcal{R}$ (resp. *functional signature* $\mathcal{F}$) is a finite set of relational symbols (resp. functional symbols), each with fixed arity. A *type symbol* is a relational symbol with arity one. A *signature* is a set of functional and relational symbols. In the sequel we use $\mathcal{R}$, resp. $\mathcal{F}$, resp. $\mathcal{T}$ for sets of relational, resp. functional, resp. type symbols.

We fix an infinite and enumerable domain $\mathbf{Dom}$ partitioned into three infinite subsets $\mathbf{Dom} = \mathbf{Iri} \cup \mathbf{Lit} \cup \mathbf{Blank}$ of IRIs, literals, and blank nodes respectively. Also, we assume an infinite subset $\mathbf{NullLit} \subseteq \mathbf{Lit}$ of null literals. In general, by *null* values we understand both null literals and blank nodes and we denote them by $\mathbf{Null} = \mathbf{NullLit} \cup \mathbf{Blank}$.

Given a signature $\mathcal{W} = \mathcal{R} \cup \mathcal{F}$, a *model* (or a *structure*) of $\mathcal{W}$ is a mapping M that with any symbol S in $\mathcal{W}$ associates its interpretation S^M s.t.:

- $R^M \subseteq \mathbf{Dom}^n$ for any relational symbol $R \in \mathcal{R}$ of arity n;
- $f^M : \mathbf{Dom}^n \to \mathbf{Dom}$, which is a total function for any function symbol $f \in \mathcal{F}$ of arity n.

We fix a countable set V of variables and reserve the symbols x, y, z for variables, and the symbols $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ for vectors of variables. We assume that the reader is familiar with the syntax of first-order logic with equality and here only recall some basic notions. A *term* over $\mathcal{F}$ is either a variable in V, or a constant in $\mathbf{Dom}$, or is of the form $f(\mathbf{x})$ where $f \in \mathcal{F}$ and the length of $\mathbf{x}$ is equal to the arity of f; we remark that we do not allow nesting of function symbols in terms. A *dependency* is a formula of the form $\forall \mathbf{x}.\varphi \Rightarrow \exists \mathbf{y}.\psi$ and in the sequel, we often drop the universal quantifier, write simply $\varphi \Rightarrow \exists \mathbf{y}.\psi$, and assume that implicitly all free variables are universally quantified.

The semantics of first-order logic formulas is captured with the *entailment* relation $M, \nu \models \phi$ defined in the standard fashion for a model M, a first-order logic formula ϕ with free variables $\mathbf{x}$ and a valuation $\nu : \mathbf{x} \to \mathbf{Dom}$. The entailment relation is extended to sets of formulas in the canonical fashion: $M \models \{\varphi_1, \ldots, \varphi_n\}$ iff $M \models \varphi_i$ for every $i \in \{1, \ldots, k\}$.

Relational Databases. We model relational databases using relational structures in the standard fashion. For our purposes we are only concerned with functional dependencies, which include key constraints. Other types of constraints, such as inclusion dependencies and foreign key constraints, are omitted in our abstraction.

A *relational schema* is a pair $\mathbf{R} = (\mathcal{R}, \Sigma_{\text{fd}})$ where $\mathcal{R}$ is a relational signature and Σ_{fd} is a set of *functional dependencies* (fds) of the form $R : X \to Y$, where $R \in \mathcal{R}$ is a relational symbol of arity n, and $X, Y \subseteq \{1, \ldots, k\}$. An fd $R : X \to Y$ is a short for the following formula $\forall \mathbf{x}, \mathbf{y}.\ R(\mathbf{x}) \wedge R(\mathbf{y}) \wedge \bigwedge_{i \in X}(x_i = y_i) \Rightarrow \bigwedge_{j \in Y}(x_j = y_j)$. An *instance* of $\mathbf{R}$ is a model I of $\mathcal{R}$ and we say that I is *valid* if $I \models \Sigma_{\text{fd}}$. The *active domain* $dom(I)$ of the instance I is the set of values from $\mathbf{Dom}$ that appear in R^I for some relational symbol R in $\mathcal{R}$. Unless we state otherwise, in the sequel we consider only instances that use only constants from $\mathbf{Lit} \setminus \mathbf{NullLit}$.

RDF Graphs and Shape Expressions Schemas. Recall that an *RDF graph*, or *graph* for short, is a set of triples in $(\mathbf{Iri} \cup \mathbf{Blank}) \times \mathbf{Iri} \times (\mathbf{Iri} \cup \mathbf{Blank} \cup \mathbf{Lit})$. The set of *nodes* of the graph G is the set of elements of $\mathbf{Iri} \cup \mathbf{Blank} \cup \mathbf{Lit}$ that appear on first or third position of a triple in G.

We next define the fragment of shape expression schemas that we consider, and that was called RBE_0 in [20]. Essentially, a ShEx is a collection of shape names, and each comes with a definition consisting of a set of triple constraints. A triple constraint indicates a label of an outgoing edge, the shape of the nodes reachable with this label, and a multiplicity indicating how many instances of this kind of edge are allowed. We remark that the constraints expressible with this fragment of ShEx, if non-recursive, can also be captured by a simple fragment of SHACL with AND operator only.

Formally, a *multiplicity* is an element of $\{1, ?, *, +\}$ with the natural interpretation: 1 is exactly one occurrence, ? stands for none or one occurrence, * stands for an arbitrary

number of occurrences, and + stands for a positive number of occurrences. A *triple constraint* over a finite set of shape names $\mathcal{T}$ is an element of $\mathbf{Iri} \times (\mathcal{T} \cup \{Lit\}) \times \{1, ?, *, +\}$, where Lit is an additional symbol used to indicate that a node is to be a literal. Typically, we shall write a triple constraint (p, T, μ) as $p :: T^{\mu}$. Now, a *shape expressions schema*, or *ShEx* schema for short, is a couple $\mathbf{S} = (\mathcal{T}, \delta)$ where $\mathcal{T}$ is a finite set of shape names, and δ is shape definition function that maps every symbol $T \in \mathcal{T}$ to a finite set of triple constraints over $\mathcal{T}$ such that for every shape name T and for every IRI p, $\delta(T)$ contains at most one triple constraint using p.

For a finite set $\mathcal{T}$ of shape names, a $\mathcal{T}$*-typed* graph is a couple $(G, typing)$ where G is a graph and $typing$ is a mapping from the nodes of G into $2^{\mathcal{T} \cup \{Lit\}}$ that with every node of G associates a (possibly empty) set of types. Let $\mathbf{S} = (\mathcal{T}, \delta)$ be a ShEx schema. The $\mathcal{T}$-typed graph $(G, typing)$ is *correctly typed* w.r.t. $\mathbf{S}$ if it satisfies the constraints defined by δ i.e., for any node n of G:

- if $Lit \in typing(n)$, then $n \in \mathbf{Lit}$;
- if $T \in typing(n)$ then $n \in \mathbf{Iri}$ and for every $p :: S^{\mu}$ in $\delta(T)$ we have that (1) for any triple (n, p, m) in G, S belongs to $typing(m)$, and (2) if K is the set of triples in G whose first element is n and second element is p, then the cardinality of K is bounded by μ i.e., $|K| = 1$ if $\mu = 1$, $|K| \leq 1$ if $\mu = ?$, and $|K| \geq 1$ if $\mu = +$ (there is no constraint if $\mu = *$).

For instance, a correct typing for the graph in Figure 2 assigns the type TBug to the nodes bug:1, bug:2, bug:3, and bug:4; the type TUser to the nodes emp:jose, user:edith, and anon:3; and Lit to every literal node.

3 ShEx Schemas as Sets of Dependencies

In this section we show how to express a ShEx schema $\mathbf{S} = (\mathcal{T}, \delta)$ using dependencies.

First, we observe that any $\mathcal{T}$-typed graph can be easily converted to a relational structure over the relational signature $\mathcal{G}_{\mathcal{T}} = \{Triple\} \cup \mathcal{T} \cup \{Lit\}$, where $Triple$ is a ternary relation symbol for encoding triples, and $\mathcal{T} \cup \{Lit\}$ are monadic relation symbols indicating node types (details in Appendix **??**). Consequently, in the sequel, we may view a $\mathcal{T}$-typed graph as the corresponding relational structure (or even a relational database over the schema $(\mathcal{G}_{\mathcal{T}}, \emptyset)$).

Next, we define auxiliary dependencies for any two $T, S \in \mathcal{T}$ and any $p \in \mathbf{Iri}$

$$tc(T, S, p) := T(x) \wedge Triple(x, p, y) \Rightarrow S(y)$$
$$mult^{\geq 1}(T, p) := T(x) \Rightarrow \exists y.\, Triple(x, p, y)$$
$$mult^{\leq 1}(T, p) := T(x) \wedge Triple(x, p, y) \wedge Triple(x, p, z) \Rightarrow y = z$$

We point out that in terms of the classical relational data exchange, tc and $mult^{\geq 1}$ are *tuple generating dependencies* (*tgd*s), and $mult^{\leq 1}$ is an *equality generating dependency* (*egd*). We capture the ShEx schema $\mathbf{S}$ with the following set of dependencies:

$$\Sigma_{\mathbf{S}} = \{tc(T, S, p) \mid T \in \mathcal{T},\ p :: S^{\mu} \in \delta(T)\} \cup$$
$$\{mult^{\geq 1}(T, p) \mid T \in \mathcal{T},\ p :: S^{\mu} \in \delta(T),\ \mu \in \{1, +\}\} \cup$$
$$\{mult^{\leq 1}(T, p) \mid T \in \mathcal{T},\ p :: S^{\mu} \in \delta(T),\ \mu \in \{1, ?\}\}.$$

Lemma 1. *For every ShEx schema* $\mathbf{S} = (\mathcal{T}, \delta)$ *and every* $\mathcal{T}$*-typed RDF graph* $(G, typing)$*,* $(G, typing)$ *is correctly typed w.r.t.* $\mathbf{S}$ *iff* $(G, typing) \models \Sigma_{\mathbf{S}}$.

4 Relational to RDF Data Exchange

In this section, we present the main definitions for data exchange.

Definition 1 (Data exchange setting). *A* relational to RDF data exchange setting *is a tuple* $\mathcal{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathcal{F}, F_{\text{int}})$ *where* $\mathbf{R} = (\mathcal{R}, \Sigma_{\text{fd}})$ *is a source relational schema,* $\mathbf{S} = (\mathcal{T}, \delta)$ *is a target ShEx schema,* $\mathcal{F}$ *is a function signature,* F_{int} *as an interpretation for* $\mathcal{F}$ *that with every function symbol* f *in* $\mathcal{F}$ *of arity* n *associates a function from* $\mathbf{Dom}^n$ *to* $\mathbf{Iri}$*, and* Σ_{st} *is a set of* source-to-target tuple generating dependencies*, clauses of the form* $\forall \mathbf{x}.\varphi \Rightarrow \psi$*, where* φ *is a conjunction of atomic formulas over the source signature* $\mathcal{R}$ *and* ψ *is a conjunction of atomic formulas over the target signature* $\mathcal{G}_{\mathcal{T}} \cup \mathcal{F}$*. Furthermore, we assume that all functions in* F_{int} *have disjoint ranges i.e., for* $f_1, f_2 \in F_{\text{int}}$ *if* $f_1 \neq f_2$*, then* $ran(f_1) \cap ran(f_2) = \emptyset$.

Definition 2 (Solution). *Take a data exchange setting* $\mathcal{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathcal{F}, F_{\text{int}})$*, and let* I *be a valid instance of* $\mathbf{R}$*. Then, a* solution *for* I *w.r.t.* $\mathcal{E}$ *is any* $\mathcal{T}$*-typed graph* J *such that* $I \cup J \cup F_{\text{int}} \models \Sigma_{\text{st}}$ *and* $J \models \Sigma_{\mathbf{S}}$.

A *homomorphism* $h : I_1 \to I_2$ between two relational structures I_1, I_2 of the same relational signature $\mathcal{R}$ is a mapping from $dom(I_1)$ to $dom(I_2)$ that 1) preserves the values of non-null elements i.e., $h(a) = a$ whenever $a \in dom(I_1) \setminus \mathbf{Null}$, and 2) for every $R \in \mathcal{R}$ and every $\mathbf{a} \in R^{I_1}$ we have $h(\mathbf{a}) \in R^{I_2}$, where $h(\mathbf{a}) = (h(a_1), \ldots, h(a_n))$ and n is the arity of R.

Definition 3 (Universal Solution). *Given a data exchange setting* $\mathcal{E}$ *and a valid source instance* I*, a solution* J *for* I *w.r.t.* $\mathcal{E}$ *is* universal*, if for any solution* J' *for* I *w.r.t.* $\mathcal{E}$ *there exists a homomorphism* $h : J \to J'$.

As usual, a solution is computed using the chase. We use a slight extension of the standard chase (explained in [**?**]) in order to handle function terms, which in our case is simple (compared to e.g. [5]) as the interpretation of function symbols is given.

5 Consistency

Definition 4 (Consistency). *A data exchange setting* $\mathcal{E}$ *is* consistent *if every valid source instance admits a solution.*

We fix a relational to RDF data exchange setting $\mathcal{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathcal{F}, F_{\text{int}})$ and let $\mathbf{S} = (\mathcal{T}, \delta)$. We normalize source-to-target tuple generating dependencies so that their right-hand-sides use exactly one $Triple$ atom and at most two type assertions on the subject and the object of the triple; such normalization is possible as our st-tgds do not use existential quantification. In this paper, we restrict our investigation to completely typed st-tgds having both type assertions, and therefore being of the following form

$$\forall \mathbf{x}.\ \varphi \Rightarrow Triple(s, p, o) \wedge T_s(s) \wedge T_o(o),$$

where s is the *subject term*, T_s is the *subject type*, $p \in \mathbf{Iri}$ is the *predicate*, o is the *object term*, and T_o is the *object type*. Because the subject of a triple cannot be a literal,

we assume that $s = f(\mathbf{y})$ for $f \in \mathcal{F}$ and for $\mathbf{y} \subseteq \mathbf{x}$, and $T_s \in \mathcal{T}$. As for the object, we have two cases: 1) the object is an IRI and then $o = g(\mathbf{z})$ for $g \in \mathcal{F}$ and for $\mathbf{z} \subseteq \mathbf{x}$, and $T_o \in \mathcal{T}$, or 2) the object is literal $o = z$ for $z \in \mathbf{x}$ and $T_o = Lit$. Moreover, we assume consistency with the target ShEx schema $\mathbf{S}$ i.e., for any st-tgd in Σ_{st} with source type T_s, predicate p, and object type T_o we have $p :: T_o^{\mu} \in \delta(T_s)$ for some multiplicity μ. Finally, we assume that every IRI constructor in $\mathcal{F}$ is used with a unique type in $\mathcal{T}$. When all these assumptions are satisfied, we say that the source-to-target tuple generating dependencies are *fully typed*.

While the st-tgds in Example 1 are not fully typed, an equivalent set of fully typed dependencies can be easily produced if additionally appropriate foreign keys are given. For instance, assuming the foreign key constraint $\textit{Bug}[uid] \subseteq \textit{User}[uid]$, the first rule with *Bug* on the left-hand-side is equivalent to

$$\textit{Bug}(b, d, u) \Rightarrow \textit{Triple}(\textit{bug2iri}(b), \mathsf{:descr}, d) \wedge \mathsf{TBug}(\textit{bug2iri}(b)) \wedge \textit{Lit}(d)$$
$$\textit{Bug}(b, d, u) \Rightarrow \textit{Triple}(\textit{bug2iri}(b), \mathsf{:rep}, \textit{pers2iri}(u)) \wedge \mathsf{TBug}(\textit{bug2iri}(b)) \wedge \mathsf{TUser}(\textit{pers2iri}(u))$$

Now, two st-tgds are *contentious* if both use the same IRI constructor f for their subjects and have the same predicate, hence the same subject type T_s and object type T_o, and $p :: T_o^{\mu} \in \delta(T_s)$ with $\mu = \mathtt{1}$ or $\mu = \mathtt{?}$. We do not want two contentious st-tgds to produce two triples with the same subject and different objects. Formally, take two contentious st-tgds σ_1 and σ_2 and assume they have the form (for $i \in \{1, 2\}$, and assuming $\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}_1, \mathbf{y}_2$ are pairwise disjoint)

$$\sigma_i = \forall \mathbf{x}_i, \mathbf{y}_i.\ \varphi_i(\mathbf{x}_i, \mathbf{y}_i) \Rightarrow \textit{Triple}(f(\mathbf{x}_i), p, o_i) \wedge T_s(f(\mathbf{x}_i)) \wedge T_o(o_i).$$

The st-tgds σ_1 and σ_2 are *functionally overlapping* if for every valid instance I of $\mathbf{R}$

$$I \cup F_{\text{int}} \models \forall \mathbf{x}_1, \mathbf{y}_1, \mathbf{x}_2, \mathbf{y}_2.\ \varphi_1(\mathbf{x}_1, \mathbf{y}_1) \wedge \varphi_2(\mathbf{x}_2, \mathbf{y}_2) \wedge \mathbf{x}_1 = \mathbf{x}_2 \Rightarrow o_1 = o_2.$$

Finally, a data-exchange setting is *key-covered* if every pair of its contentious st-tgds is functionally overlapping. Note that any single st-tgd may be contentious with itself.

Theorem 1. *A fully typed data exchange setting is consistent if and only if it is key-covered.*

The sole reason for the non-existence of a solution for a source instance I is a violation of some egd in $\Sigma_{\mathbf{S}}$. The key-covered property ensures that such egd would never be applicable. Intuitively, two egd-conflicting objects o_1 and o_2 are necessarily generated by two contentious st-tgds. The functional-overlapping criterion guarantees that the terms o_1 and o_2 are "guarded" by a primary key in the source schema, thus cannot be different.

Theorem 2. *It is decidable whether a fully typed data exchange setting is key-covered.*

The proof uses a reduction to the problem of functional dependency propagation [12].

6 Universal Solution

In this section, we identify conditions that guarantee the existence of a universal solution. Our results rely on the existence of a universal solution for sets of weakly acyclic

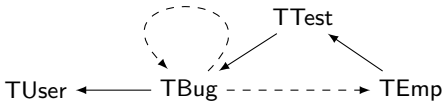

Fig. 3: Dependency graph with dashed weak edges and plain strong edges

sets of dependencies for relational data exchange [10]. As the tgds and egds that we generate are driven by the schema (cf. Section 3), we introduce a restriction on the ShEx schema that yields weakly acyclic sets of dependencies, and consequently, guarantees the existence of universal solution.

The *dependency graph* of a ShEx schema $\mathbf{S} = (\mathcal{T}, \delta)$ is the directed graph whose set of nodes is $\mathcal{T}$ and has an edge (T, T') if T' appears in some triple constraint $p :: T'^{\mu}$ of $\delta(T)$. There are two kinds of edges: *strong edge*, when the multiplicity $\mu \in \{1, +\}$, and *weak edge*, when $\mu \in \{*, ?\}$. The schema $\mathbf{S}$ is *strongly recursive* if its dependency graph contains a cycle of strong edges only, and is *weakly recursive* otherwise. Take for instance the following extension of the ShEx schema from Example 1:

$$\mathsf{TUser} \to \{\mathsf{:name} :: Lit^{1}, \mathsf{:email} :: Lit^{1}, \mathsf{:phone} :: Lit^{?}\}$$
$$\mathsf{TBug} \to \{\mathsf{:rep} :: \mathsf{TUser}^{1}, \mathsf{:descr} :: Lit^{1}, \mathsf{:related} :: \mathsf{TBug}^{*}, \mathsf{:repro} :: \mathsf{TEmp}^{?}\}$$
$$\mathsf{TEmp} \to \{\mathsf{:name} :: Lit^{1}, \mathsf{:prepare} :: \mathsf{TTest}^{+}\}$$
$$\mathsf{TTest} \to \{\mathsf{:covers} :: \mathsf{TBug}^{+}\}$$

The dependency graph of this schema, presented in Figure 3. contains two cycles but neither of them is strong. Consequently, the schema is weakly recursive (and naturally so is the ShEx schema in Example 1).

As stated above, a weakly-recursive ShEx schema guarantees a weakly-acyclic set of dependencies and using results from [10] we get

Proposition 1. *Let $\mathcal{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\mathrm{st}}, \mathcal{F}, F_{\mathrm{int}})$ be a data exchange setting and I be a valid instance of $\mathbf{R}$. If $\mathbf{S}$ is weakly recursive, then every chase sequence of I with $\Sigma_{\mathrm{st}} \cup \Sigma_{\mathbf{S}}$ is finite, and either every chase sequence of I with Σ_{st} fails, or every such chase sequence computes a universal solution of I for $\mathcal{E}$.*

7 Conclusion and Future Work

We presented a preliminary study of the consistency problem for relational to RDF data exchange in which the target schema is ShEx. Consistency is achieved by fully typed and key-covered syntactic restriction of st-tgds. An open problem that we plan to investigate is consistency when the fully typed restriction is relaxed; we believe that it is achievable if we extend the definition of contentious st-tgds. Another direction of research is to consider a larger subset of ShEx. Finally, we plan to extend our framework to typed literals which are not expected to bring fundamental difficulties but are essential for practical applications.

Acknowledgments This work was partially supported by a grant from CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020 and by the ANR project DataCert ANR-15-CE39-0009.

References

1. R2RML: RDB to RDF Mapping Language. W3C Recommendation 27 September 2012, http://www.w3.org/TR/r2rml/
2. RDF 1.1 Semantics. W3C Recommendation 25 February 2014, https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/
3. Shapes Constraint Language (SHACL). W3C Recommendation 20 July 2017, https://www.w3.org/TR/shacl/
4. Arenas, M., Barcelo, P., Libkin, L., Murlak, F.: Relational and XML Data Exchange. Morgan and Claypool Publishers (2010)
5. Arenas, M., Pérez, J., Reutter, J., Riveros, C.: The Language of Plain SO-tgds: Composition, Inversion and Structural Properties. J. Comput. Syst. Sci. (2013)
6. Beeri, C., Vardi, M.Y.: The implication problem for data dependencies. In: Even, S., Kariv, O. (eds.) Automata, Languages and Programming (1981)
7. Boneva, I., Bonifati, A., Ciucanu, R.: Graph Data Exchange with Target Constraints. In: EDBT/ICDT Workshops - Querying Graph Structured Data (GraphQ) (2015)
8. Boneva, I., Labra Gayo, J.E., Prud'hommeaux, E.G.: Semantics and Validation of Shapes Schemas for RDF. In: International Semantic Web Conference (2017)
9. Fagin, R., Haas, L.M., Hernández, M.A., Miller, R.J., Popa, L., Velegrakis, Y.: Clio: Schema Mapping Creation and Data Exchange. In: Conceptual Modeling: Foundations (2009)
10. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theoretical Computer Science (2005)
11. Fuxman, A., Hernández, M.A., Ho, C.T.H., Miller, R.J., Papotti, P., Popa, L.: Nested Mappings: Schema Mapping Reloaded. In VLDB pp. 67–78 (2006)
12. Klug, A., Price, R.: Determining View Dependencies Using Tableaux. ACM Trans. Database Syst. (1982)
13. Kolaitis, P.G., Panttaja, J., Tan, W.C.: The complexity of data exchange. In: Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 30–39 (2006)
14. Köpcke, H., Rahm, E.: Frameworks for Entity Matching: A Comparison. Data Knowl. Eng. (2010)
15. Marnette, B., Geerts, F.: Static Analysis of Schema-mappings Ensuring Oblivious Termination. In: Proceedings of the International Conference on Database Theory (2010)
16. Marnette, B., Mecca, G., Papotti, P., Raunich, S., Santoro, D., Roma, U.R.T.: ++Spicy: an Open-Source Tool for Second-Generation Schema Mapping and Data Exchange (2011)
17. Ryman, A., Hors, A.L., Speicher, S.: Oslc resource shape: A language for defining constraints on linked data. In: Workshop on Linked Data on the Web (2013)
18. Sequeda, J.F., Arenas, M., Miranker, D.P.: On Directly Mapping Relational Databases to RDF and OWL. In: Proceedings of the 21st International Conference on World Wide Web (2012)
19. Sirin, E.: Data Validation with OWL Integrity Constraints. In: Hitzler, P., Lukasiewicz, T. (eds.) Web Reasoning and Rule Systems. pp. 18–22 (2010)
20. Staworko, S., Boneva, I., Labra Gayo, J.E., Hym, S., Prud'hommeaux, E.G., Solbrig, H.R.: Complexity and Expressiveness of ShEx for RDF. In: ICDT (2015)