Consistency and Certain Answers in Relational to RDF Data Exchange with Shape Constraints

Iovka Boneva, Slawek Staworko, Jose Martin Lozano Aparicio

Consistency and Certain Answers in Relational to RDF Data Exchange with Shape Constraints

Iovka Boneva, Sławek Staworko, and Jose Lozano

Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France

Abstract. We investigate the data exchange from relational databases to RDF graphs inspired by R2RML with the addition of target shape schemas capturing fragments of SHACL and ShEx. We study the problems of *consistency* i.e., checking that every source instance admits a solution, and *certain query answering* i.e., finding answers present in every solution. We identify the class of *constructive relational to RDF data exchange* that uses IRI constructors and full tgds (with no existential variables) in its source to target dependencies. We show that the consistency problem is coNP-complete. We introduce the notion of *universal simulation solution* that allows to compute certain query answers to any class of queries that is *robust under simulation*. One such class are nested regular expressions (NREs) that are *forward* i.e., do not use the inverse operation. Using universal simulation solution renders tractable the computation of certain answers to forward NREs (data-complexity).

1 Introduction

The recent decade has seen RDF raise to the task of interchanging data between Web applications [25]. In many applications the data is stored in a relational database and only exported as RDF, as evidenced by the proliferation of languages for mapping relational databases to RDF, such as R2RML [18], Direct Mapping [4] or YARRRML [20]. As an example, consider the following R2RML mapping, itself in RDF format presented in Turtle syntax

```
<#EmpMap>
  rr:logicalTable [ rr:sqlQuery "SELECT id, name, email FROM Emp NATURAL JOIN Email" ];
  rr:subjectMap [ rr:template "emp:{id}"; rdf:type :TEmp ];
  rr:predicateObjectMap [ rr:predicate :name; rr:objectMap [ rr:column "name"] ];
  rr:predicateObjectMap [ rr:predicate :email; rr:objectMap [ rr:column "email"] ].
```

It exports the join of two relations $Emp(\underline{id}, name)$ and $Email(\underline{id}, name)$ into a set of triples. For every employee it creates a dedicated Internationalized Resource Identifier (IRI) consisting of the prefix `emp:` and the employee identifier. More importantly, the class (`rdf:type`) of each employee IRI is declared as :TEmp.

RDF has been originally proposed schema-less to promote its adoption but the need for schema languages for RDF has been since identified and deemed particularly important in the context of exchange of data between applications [23,32]. One family of proposed schema formalisms for RDF is based on *shape constraints*

and this class includes *shape constraint language* (SHACL) [17,22] and *shape expressions schemas* (ShEx) [11,27,28]. The two languages allow to define a set of types that impose structural constraints on nodes and their immediate neighborhood in an RDF graph. For instance, the type :TEmp may be defined as

:TEmp { :name xsd:string; :email xsd:string?; :works @:TDept+ }

Essentially, an employee IRI must have a single :name property, an optional :email property that are both strings, and at least one :works property each leading to an IRI of type :TDept.

In the present paper we formalize the process of exporting a relational database to RDF as *data exchange*, and study two of its fundamental problems: *consistency* and *certain query answering*. In data exchange the mappings from the source database to the target database are modeled with *source-to-target tuple-generating dependencies* (st-tgds). For mappings defined with R2RML we propose a class of *full constructive* st-tgds, which use *IRI constructors* to map entities from the relational database to IRIs in the RDF. For instance, the R2RML mapping presented before can be expressed with the following st-tgd

$$\begin{aligned} Emp(id, name) \wedge Email(id, email) \Rightarrow\ & Triple(emp2iri(id), :\mathsf{name}, name) \wedge \\ & Triple(emp2iri(id), :\mathsf{email}, email) \wedge \\ & \mathsf{TEmp}(emp2iri(id)), \end{aligned}$$

where *emp2iri* is an IRI constructor that generates an IRI for each employee. To isolate the concerns, in our analysis of the st-tgds we refrain form inspecting the definitions of IRI constructors and require only that they are *non-overlapping*, i.e. no two IRI constructors are allowed to output the same IRI. We call the above setting *constructive relational to RDF data exchange*. We report that in this setting all 4 use cases of R2RML [6] can be expressed. Furthermore, we can cover 38 out of 54 test cases for R2RML implementations [31]. Among the non-covered test cases, 9 use pattern-based function to transform data values and 7 use SQL statements with aggregation functions. In fact, our assessment is that the proposed framework allows to fully address all but one out of the 11 core functional requirements for R2RML [6], namely the *Apply a Function before Mapping*. Finally, in our investigations we consider a class of *shape schemas* that are at the intersection of SHACL and ShEx. They are known to have desirable computational properties while remaining practical, and furthermore, they posses a sought-after feature of having an equivalent graphical representation (in the form of shape graphs) [29].

For a given source relational instance, a *solution* to data exchange is a target database (an RDF graph in our case) that satisfies the given set of st-tgds and the target schema (a shape schema in our case). The number of solutions may vary from none to infinitely many. The problem of *consistency* is motivated by the need for static verification tools that aim to identify potentially erroneous data exchange settings that are *inconsistent* i.e., admit no solution for some source database instance. In general, a consistent data exchange setting may yield many

solutions to a given source instance and it is commonplace to apply the *possible world semantics* [21] to evaluate queries: a *certain answer* is an answer returned in every solution. It is standard practice to construct a solution that allows to easily compute certain answers. In the case of relational data exchange, *universal solutions* have been identified and allow to easily compute certain answers to conjunctive queries, or any class of queries preserved under homomorphism for that matter [19]. Unfortunately, for relational to RDF data exchange with target shape schema, a finite universal solution might not exists, even if the setting is consistent and admits solutions. Also, the class of conjunctive queries, while adequate for expressing queries for relational databases, is less so for RDF. Query languages, like SPARQL, allowing regular path expressions with nesting have been proposed to better suit the needs of querying RDF [26].

Our contributions are as follows. We formalize the framework of relational to RDF data exchange with target shape schema and IRI constructors, and we identify the class of *constructive relational to RDF data exchange* that uses shape schemas and full constructive source-to-target dependencies. We provide an effective characterization of consistency of constructive relational to RDF data exchange settings and show that the problem is coNP-complete. We propose a novel notion of *universal simulation solution*, that can be constructed for any consistent constructive relational to RDF data exchange setting, and use it to show tractability of computing certain answers to *forward nested regular expressions*. In an extended version of the present paper [14] we present full details and study further a number of extensions of our framework and show each time negative computational consequences.

2 Preliminaries

In this section we recall the standard notions of logic and databases [1,24].

Relational databases A *relational schema* is a pair $\mathbf{R} = (\mathscr{R}, \Sigma_{\text{fd}})$ where $\mathscr{R}$ is a set of relation names and Σ_{fd} is a set of functional dependencies. Each relation name has a fixed arity and a set of attribute names. A *functional dependency* is written as usual $R : X \to Y$ where R is a relation name and X and Y are two sets of attributes of R. An *instance* I of $\mathbf{R}$ is a function that maps every relation name of $\mathbf{R}$ to a set of tuples over a set Lit of constants (also called literal values). The instance I is *consistent* if it satisfies all functional dependencies Σ_{fd}. In the sequel, we often view an instance as a relational structure over the signature $\mathscr{R}$.

Graphs An RDF graph G is a labeled graph whose nodes are divided into two *kinds*: *literal* nodes (Lit) and *non-literal* (Iri) nodes with only the latter allowed to have outgoing edges. Every node is labeled and we adopt the *unique name assumption* (UNA) i.e., no two node have the same label. Consequently, we equate nodes with their labels and by $nodes(G)$ we denote the set of labels of nodes of G. Also, each edge is labeled with a predicate name, which is a non-null resource name Pred. As a result of using chase some nodes may be labeled with names nulls.

Shape Schemas A *shapes schema* is a pair $\mathbf{S} = (\mathscr{T}, \delta)$, where $\mathscr{T}$ is a finite set of *type names* and $\delta : \mathscr{T} \times \mathsf{Pred} \to (\mathscr{T} \cup \{Literal\}) \times \{\mathtt{1}, \mathtt{?}, \mathtt{*}, \mathtt{+}\}$ defines shape constraints. A *shape constraint* $\delta(T, p) = (S, \mu)$, often presented as $\delta(T, p) = S^{\mu}$, reads as follows: if a node has type T, then every neighbor reached with an outgoing p-edge must have type S and the number of such neighbors must be within the bounds of μ: precisely one if $\mu = \mathtt{1}$, at most one if $\mu = \mathtt{?}$, at least one if $\mu = \mathtt{+}$, and arbitrarily many if $\mu = \mathtt{*}$. Whenever $\mu = \mathtt{1}$ or $\mu = \mathtt{?}$ we say that the predicate p is *functional* for type T.

Dependencies We employ the standard syntax of first-order logic (cf. [24]). In the sequel, we shall view graphs as relational structures using the ternary predicate *Triple* and monadic predicates in $\mathscr{T} \cup \{Literal\}$ to indicate the types of nodes. Furthermore, in formulas we use the edge labels Pred as constant symbols. Later on, we additionally introduce functions that allow to map the values in relational databases to resource names used in RDF graphs, and we allow the use of function names in formulas but without nesting.

Now, a *dependency* is a formula of the form $\forall \bar{x}.\varphi \Rightarrow \exists \bar{y}.\psi$, where φ is called the *body* and ψ the *head* of the dependency, and we typically omit the universally quantified variables and write simply $\varphi \Rightarrow \exists \bar{y}.\psi$. A dependency is *equality-generating* (egd) if its body is a clause and its head consists of an equality condition $x = y$ on a pair of variables. A *tuple-generating dependency* (tgd) uses clauses in both its head and its body. A tgd is *full* if it has no existentially quantified variables.

A number of previously introduced concepts can be expressed with dependencies. Any functional dependency is in fact an equality-generating dependency. Interestingly, any deterministic shape schema $\mathbf{S}$ can be expressed with a set $\Sigma_{\mathbf{S}}$ of equality- and tuple-generating dependencies. More precisely, whenever $\delta(T, p) = S^{\mu}$ the set $\Sigma_{\mathbf{S}}$ contains:

(TP) the *type propagation* rule: $T(x) \wedge \mathit{Triple}(x, p, y) \Rightarrow S(y)$,
(PF) the *predicate functionality* rule if $\mu = \mathtt{1}$ or $\mu = \mathtt{?}$:
$T(x) \wedge \mathit{Triple}(x, p, y_1) \wedge \mathit{Triple}(x, p, y_2) \Rightarrow y_1 = y_2$,
(PE) the *predicate existence* rule if $\mu = \mathtt{1}$ or $\mu = \mathtt{+}$: $T(x) \Rightarrow \exists y.\ \mathit{Triple}(x, p, y)$.

3 Constructive Relational to RDF Data Exchange

An n-ary *IRI constructor* is a function $f : \mathsf{Lit}^n \to \mathsf{Iri}$ that maps an n-tuple of database constants to an RDF resource name. A *IRI constructor* library is a pair $\mathbf{F} = (\mathscr{F}, F)$, where $\mathscr{F}$ is a set of IRI constructor names and F is their interpretation. $\mathbf{F}$ is *non-overlapping* if all its IRI constructors have pairwise disjoint ranges.

Definition 1. A *relational to RDF data exchange setting with fixed IRI constructors* is a tuple $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\mathrm{st}}, \mathbf{F})$, where $\mathbf{R} = (\mathscr{R}, \Sigma_{\mathrm{fd}})$ is a source relational schema, $\mathbf{S} = (\mathscr{T}, \delta)$ is a target shape constraint schema, $\mathbf{F} = (\mathscr{F}, F)$ is an IRI constructor library, and Σ_{st} is a set of *source-to-target tuple generating dependencies (*st-tgds*)* whose bodies are formulas over $\mathscr{R}$ and heads are formulas over

$\mathscr{F} \cup \mathscr{T} \cup \{Literal\}$ without nesting of function symbols in $\mathscr{F}$. $\mathscr{E}$ is *constructive* if the library of IRI constructors is non-overlapping and the st-tgds Σ_{st} are full tgds. A typed graph J is a *solution* to $\mathscr{E}$ for a source instance I of $\mathbf{R}$, iff J satisfies $\mathbf{S}$ and $I \cup J \cup F \models \Sigma_{\text{st}}$. By $sol_{\mathscr{E}}(I)$ we denote the set of all solutions for I to $\mathscr{E}$. □

In the reminder we fix a constructive data exchange setting $\mathscr{E}$, and in particular, we assume a fixed library of IRI constructors $\mathbf{F}$. Since we work only with constructive data exchange settings, w.l.o.g. we can assume that the heads of all st-tgds consist of one atom only. We point out that while a constructive data exchange setting does not use egds, our constructions need to accommodate egds and tgds coming from the shapes schema.

The standard *chase* procedure allows to construct a solution to $\mathscr{E}$ for a source instance I. However, such solution might not exist either because the chase fails due to an unsatisfiable egd, or because it never terminates. The *core pre-solution* for I to $\mathscr{E}$ is the result J_0 of chase on I with the st-tgds Σ_{st} and all **TP** rules of $\mathbf{S}$. In essence J_0 is obtained by exporting the relational data to RDF triples with Σ_{st} and then propagating any missing types according to $\mathbf{S}$ but without creating any new nodes with **PE** rules. This process does not introduce any null values and always terminates yielding a unique result. Naturally, J_0 is included in any solution $J \in sol_{\mathscr{E}}(I)$.

4 Consistency

Recall that a data exchange setting $\mathscr{E}$ is *consistent* if every consistent source instance I of $\mathbf{R}$ admits a solution to $\mathscr{E}$.[1] Throughout this section we fix a data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$ and study its consistency.

Note that only predicate functionality egds can bring inconsistency. We show that $\mathscr{E}$ is inconsistent if and only if there is a source instance I and a predicate functionality egd $T(x) \wedge Triple(x, p, y) \wedge Triple(x, p, y') \Rightarrow y = y'$ in $\mathscr{E}$ such that all solutions to $\mathscr{E}$ for I need to satisfy the body of the above rule but y and y' are not equatable. We distinguish two situations, that together provide a necessary and sufficient condition for $\mathscr{E}$ to be (in-)consistent.

- *value inconsistency* when y and y' are different constants (see Section 4),
- *node kind inconsistency* when one among y, y' is a literal and the other is not (see Section 4).

Value Consistency Let $\Sigma_{\mathbf{S}}^{\mathbf{PF}}$ be the set of predicate functionality rules from $\Sigma_{\mathbf{S}}$ as defined in Section 3.

An instance I of $\mathbf{R}$ is called *value inconsistent* if $J \not\models \Sigma_{\mathbf{S}}^{\mathbf{PF}}$, where J is the core pre-solution for I to $\mathscr{E}$. The data exchange setting $\mathscr{E}$ is called *value inconsistent* if there exists a consistent instance I of $\mathbf{R}$ that is value inconsistent.

We now sketch a decision procedure *value-inconsistent*($\mathscr{E}$) that tests whether $\mathscr{E}$ is value inconsistent by constructing a value inconsistent source instance

[1] This was called absolute consistency in [9].

whenever such exists. It is illustrated on the following example data exchange setting. The relational signature contains symbols R, S both of arity two, and the IRI constructors are $\{g_0, g, f\}$ all of arity one. The shapes schema $\mathbf{S}$ is given by $\delta(U_0, r) = U^*$, $\delta(U, q) = T^*$, $\delta(T, p) = \mathit{Literal}^1$, and the st-tgds are:

$$(\sigma_1)\ R(x_0, x_1) \Rightarrow U_0(g_0(x_1)) \qquad R(x, z) \wedge S(x, y') \Rightarrow \mathit{Triple}(f(x'), p, y')\ (\sigma)$$
$$(\sigma_2)\ R(x_1, x_2) \Rightarrow \mathit{Triple}(g_0(x_1), r, g(x_2)) \qquad S(x, y) \Rightarrow \mathit{Triple}(f(x), p, y)\ (\sigma')$$
$$(\sigma_3)\ R(x_2, x) \Rightarrow \mathit{Triple}(g(x_2), q, f(x))$$

Using that IRI constructors have pairwise disjoint ranges, it follows from the definition that a source instance is value inconsistent iff its core pre-solution contains a triple of facts of the form $\{T(f(x)), \mathit{Triple}(f(x), p, b), \mathit{Triple}(f(x), p, b')\}$ for some function symbol f, and such that x and $y \neq y'$ are constants and the predicate p is functional for type T in $\mathbf{S}$. We call such triple of facts a *violation* and (T, p, f) its *sort*. There is a finite number of possible violation sorts for $\mathscr{E}$. On the example, the possible violation sorts are $(T, p, f), (T, p, g_0)$ and (T, p, g), as the unique functional predicate rule is for type T and predicate p. The procedure *value-inconsistent*$(\mathscr{E})$ enumerates all violation sorts and for each of them tries to build a value inconsistent instance.

So consider the violation sort (T, p, f). Although none of the st-tgds heads contains a fact of the form $T(f(\_))$, we remark that such fact can be obtained using type propagation rules. Indeed, consider the source instance $I' = \{R(x_0, x_1), R(x_1, x_2), R(x_2, x)\}$ obtained as the union of the bodies of st-tgds (σ_1), (σ_2) and (σ_3), where variables are used as constants. Applying on I' the st-tgds (σ_1), (σ_2) and (σ_3) together with the type propagation rules for U_0, r and U, q yields the target instance $J' = I' \cup \{U_0(g_0(x_1)), \mathit{Triple}(g_0(x_1), r, g(x_2)), U(g(x_2)), \mathit{Triple}(g(x_2), q, f(x)), T(f(x))\}$ which does contain a fact $T(f(x))$ as required. We show that for given T and f, a fact of the form $T(f(\_))$ exists in some core pre-solution w.r.t. $\mathscr{E}$ if and only if $\mathscr{E}$ contains an appropriate finite and elementary (i.e. without repetitions) sequence of st-tgds such as (σ_1), (σ_2),(σ_3) above that, combined with type propagation rules, allows to obtain $T(f(\_))$.

Now we need to add facts of the forms $\mathit{Triple}(f(x), p, y)$ and $\mathit{Triple}(f(x), p, y')$. This is done using the st-tgds (σ) and (σ'), called *contentious* for f and p. So consider $I = I' \cup \{S(x, y), R(x, z), S(x, y')\}$ obtained by adding the bodies of (σ) and (σ') to I'. Its core pre-solution is $J = J' \cup \{\mathit{Triple}(f(x), p, y), \mathit{Triple}(f(x'), p, y'), \mathit{Literal}(y), \mathit{Literal}(y')\}$ and it contains a violation of sort (T, p, f) whenever $y \neq y'$.

Thus, I is an instance over the source signature that is value inconsistent. This does not imply yet that $\mathscr{E}$ is inconsistent, as I might not be consistent w.r.t the source functional dependencies. If the first attribute of S is a primary key, then $y = y'$, I is not value inconsistent and we can actually show that the example data exchange setting $\mathscr{E}$ is consistent. Otherwise, I is value inconsistent, and so is $\mathscr{E}$.

Here is a NP procedure that checks value inconsistency by guessing an inconsistent source instance:

- guess a violation sort (T, p, f) of $\mathscr{E}$,

- guess an elementary sequence of st-tgds $\sigma_1, \ldots, \sigma_n$ that allows to generate a fact $T(f(\_))$,
- guess two st-tgds σ, σ' contentious for f and p,
- construct in PTIME a source instance I as the union of the bodies of $\sigma_1, \ldots, \sigma_n, \sigma, \sigma'$ (after appropriate renaming of variables),
- show in PTIME that I satisfies the source functional dependencies.

Theorem 1. *Value consistency of a data exchange setting is in coNP.*

Node Kind Consistency Node kind inconsistency is specific to relational to RDF data exchange due to the presence of two types of values, namely IRIs and literals. It corresponds to situations in which two values are equated by a predicate functionality rule while one of them is a literal (that is, has type *Literal*) but the other is not (that is, has a type $T \in \mathscr{T}$). Therefore we identify node kind inconsistency with the fact that a node has both types *Literal* and $T \neq \mathit{Literal}$.

For a typed graph J and a node $n \in \mathit{nodes}(J)$, let $\mathit{types}_J(n) = \{T \in \mathscr{T} \cup \{\mathit{Literal}\} \mid T(n) \in J\}$. For a source instance I, define its sets of *co-occurring types* as $\mathit{CoTypes}(I) = \{\mathit{types}_J(n) \mid J \in \mathit{sol}_{\mathscr{E}}(I),\ n \in \mathit{nodes}(J)\}$. Let $\mathit{CoTypes}(\mathscr{E}) = \bigcup_{\text{I instance of } \mathbf{R}} \mathit{CoTypes}(I)$.

A source instance I is called *node kind inconsistent* if $\mathit{CoTypes}(I)$ contains a set X s.t. $\{\mathit{Literal}, T\} \subseteq X$ for some T in $\mathscr{T}$. The data exchange setting $\mathscr{E}$ is called node kind inconsistent if there is a node kind inconsistent instance I of $\mathbf{R}$.

We show that $\mathscr{E}$ is node kind inconsistent iff $\mathit{CoTypes}(\mathscr{E})$ contains a set X such that $\{\mathit{Literal}, T\} \subseteq X$ for some T in $\mathscr{T}$. Furthermore, for any $X \subseteq \mathscr{T} \cup \{\mathit{Literal}\}$, we can test in PTIME whether X belongs to $\mathit{CoTypes}(\mathscr{E})$. Therefore,

Proposition 1. *Checking node kind inconsistency of $\mathscr{E}$ is in NP.*

The central claim of this section is Theorems 2 below. The lower bound is shown using a reduction to the complement of SAT.

Theorem 2. *A constructive relational to RDF data exchange setting $\mathscr{E}$ is consistent iff it is value consistent and node kind consistent. Checking consistency of a constructive relational to data exchange setting is decidable and coNP-complete.*

5 Certain Query Answering

In this section we investigate computing certain answers to Boolean queries focusing on a subclass of nested regular expressions (NREs). In [14] we show that our results extend to non-Boolean queries but also that handling the full class of NREs leads to an increase in computational complexity.

Throughout this section we fix a constructive data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\mathrm{st}}, \mathbf{F})$ and assume $\mathscr{E}$ is consistent. We recall that for a Boolean graph query Q, *true* is the *certain answer* to Q in I w.r.t. $\mathscr{E}$ iff *true* is the answer to Q in every solution to $\mathscr{E}$ for I.

The standard approach to computing certain answers is to construct a universal solution with the chase and evaluate the query against it (and to drop any answers with null values) [19]. However, in our case a finite universal solution may not exist because the chase may enter an infinite loop due to **PE** rules when the shape schema is strongly-recursive i.e., it has a cycle with multiplicities of `1` and `+`. Infinite chase corresponds to an attempt to unravel such cycles by inventing new nodes ad infinitum. Instead, we construct a solution where a new node is invented only if one satisfying precisely the same types has not been invented before. Such a solution is not universal, but interestingly, it has a different flavor of universality, one that can be captured with the standard notion of graph simulation: any solution can be simulated in it. We also show that this notion of universality is good enough for classes of queries that are robust under simulation, and we identify a practical class of forward nested regular expressions with this property. This yields a practical class of queries with tractable certain answers.

Nested regular expressions In this paper we work with the class of *nested regular expressions* (NREs) that have been proposed as the navigational core of SPARQL [26]. In essence, NREs are regular expressions that use concatenation $\cdot$, union $+$, Kleene's closure $*$, inverse $-$, and permit nesting and testing node and edge labels. We refer the reader to [14] for detailed definition. We point out that NREs are incompatible with conjunctive queries but even forward NREs capture the subclass of acyclic conjunctive queries. Also, forward NREs properly capture regular path queries.

Graph simulation and robust query classes We adapt the classic notion of graph simulation to account for null values. Formally, a *simulation* of a graph G by a graph H is a relation $R \subseteq \mathit{nodes}(G) \times \mathit{nodes}(H)$ such that for any $(n, m) \in R$, we have 1) n is a literal node if and only if m is a literal node, 2) if n is not null, then m is not null and $n = m$, and 3) for any outgoing edge from n with label p that leads to n' there is a corresponding outgoing edge from m with label p that leads to m' such that $(n', m') \in R$. The set of simulations is closed under union, and consequently, there is always one maximal simulation, and if (n, m) is contained in it, we say that n is simulated by m. Also, we say that G is *simulated* by H if every node of G is simulated by a node of H. We are interested in simulations because they capture the essence of exploring a graph by means of following outgoing edges only.

Definition 2. A class $\mathcal{Q}$ of Boolean queries on graphs is *robust under simulation* iff for any query $Q \in \mathcal{Q}$ and any two graph G and H such that G is simulated by H, if Q is true in G, then Q is true in H. □

Naturally, the class of forward NREs has this very property.

Lemma 1. *The class of forward nested regular expressions is robust under simulation.*

Universal simulation solution When dealing with classes of queries that are robust under simulation we employ simulation instead of homomorphism to define a solution that allows to find all certain answers.

Definition 3. A typed graph $\mathscr{U}$ is a *universal simulation solution* to $\mathscr{E}$ for I iff $\mathscr{U}$ is simulated by every solution J to $\mathscr{E}$ for I. □

And indeed, a universal simulation solution does allow us to capture certain answers for queries from classes robust under simulation.

Theorem 3. *Let $\mathscr{Q}$ be a class of Boolean graph queries robust under simulation. For any query $Q \in \mathscr{Q}$ and any consistent instance I of $\mathbf{R}$, true is the certain answer to Q in I w.r.t. $\mathscr{E}$ if and only if true is the answer to Q in a universal simulation solution to $\mathscr{E}$ for I.*

The main challenge is in constructing a universal simulation solution. The precise construction is presented in [14] and we outline it roughly. First we begin with the core pre-solution that is obtained from the source instance I with the the st-tgds Σ_{st} and the **TP** rules for $\mathbf{S}$ that propagate the types according to the shape schema. Then, we add fresh null values that ensure satisfaction of the schema, as required by the **PE** rules for $\mathbf{S}$. We point out that each null node corresponds to a subset of types of $\mathbf{S}$ that it needs to satisfy, which bounds their number by $2^{|\Gamma|}$, and furthermore, using the Chinese reminder theorem we show that this bound is tight. To ensure that the produced universal simulation solution has the smallest size, we employ the standard technique of quotient by bisimulation of the obtained graph [30].

Theorem 4. *For an instance I of $\mathbf{R}$, we can construct a size-minimal universal simulation solution $\mathscr{U}_0$ in time polynomial in the size of I and exponential in the size of $\mathbf{S}$. The size of $\mathscr{U}$ is bounded by a polynomial in the size of I and an exponential function in the size of $\mathbf{S}$.*

Complexity We can now characterize the data complexity of certain query answering. Recall that data complexity assumes the query and the data exchange setting to be fixed, and thus of fixed size, and only the source instance is given on the input. Consequently, the size of universal simulation solution $\mathscr{U}_0$ is polynomially-bounded by the size of I. Since the data complexity of evaluating NREs is know to be PTIME [26], we get the following result.

Theorem 5. *The data complexity of computing certain answers to forward nested regular expressions w.r.t. constructive relational to RDF data exchange setting is in PTIME.*

6 Related Work and Conclusions

R2RML is a W3C standard language for defining custom relational to RDF mappings [18], other languages such as YARRRML [20] are compiled to R2RML but they do not consider target constraints. Data exchange has been considered for graph databases with varying expressive power of mapping formalisms such as nested regular expressions [7,10], which is however incomparable with shape schemas. In [13] we have demonstrated a graphical tool for defining constructive

relational to RDF mappings. Our results do not follow from the exists results on the standard relational data exchange [19], which are either too limited in their expressive power [15,19] or come with a significant complexity penalty [2,9,3,5,8,16]

We have presented a data exchange framework for modeling R2RML scripts, we have studied the problems of consistency and certain query answering, and characterized their complexity. In [14] we also show that extending the framework in a number of natural directions generally leads to an increase of complexity.

References

1. S. Abiteboul, R. Hull, and V Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. S. Amano, C. David, L. Libkin, and F. Murlak. XML schema mappings: Data exchange and metadata management. *J. ACM*, 61(2):12:1–12:48, 2014.
3. M. Arenas, P. Barceló, and J. L. Reutter. Query languages for data exchange: Beyond unions of conjunctive queries. *Theory Comput. Syst.*, 49(2):489–564, 2011.
4. M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda. A Direct Mapping of relational data to RDF. W3C Recomm., 2012.
5. M. Arenas and L. Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2):7:1–7:72, 2008.
6. S. Auer, L. Feigenbaum, D. Miranker, A. Fogarolli, and J. Sequeda. Use cases and requirements for mapping relational databases to RDF, 2010. W3C.
7. P. Barceló, J. Pérez and J. L. Reutter. Schema mappings and data exchange for graph databases. In *International Conference on Database Theory (ICDT)*, pages 189–200, 2013.
8. M. Bienvenu, M. Ortiz, and M. Simkus. Regular path queries in lightweight description logics: Complexity and algorithms. *J. Artif. Intell. Res.*, 53:315–374, 2015.
9. M. Bojańczyk, L. A. Kołodziejczyk, and F. Murlak. Solutions in XML data exchange. *J. Comput. Syst. Sci.*, 79(6):785–815, 2013.
10. I. Boneva, A. Bonifati, and R. Ciucanu. Graph data exchange with target constraints. In *EDBT/ICDT Workshops (GraphQ)*, pages 171–176, 2015.
11. I. Boneva, J. E. Labra Gayo, and E. G. Prud'hommeaux. Semantics and validation of shapes schemas for RDF. In *International Semantic Web Conference (ISWC)*, pages 104–120, 2017.
12. I. Boneva, J. Lozano, and S. Staworko. Relational to RDF data exchange in presence of a shape expression schema. In *Alberto Mendelzon International Workshop (AMW)*, 2018.
13. I. Boneva, J. Lozano, and S. Staworko. ShERML: Mapping relational data to RDF. In *ISWC Satellite Tracks*, pages 213–216, 2019.
14. I. Boneva, S. Staworko, and J. Lozano. Consistency and certain answers in relational to RDF data exchange with shape constraints. Technical report, arXiv:2003.13831, April 2020.
15. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Semant.*, 14:57–83, 2012.
16. D. Calvanese, T. Eiter, and M. Ortiz. Answering regular path queries in expressive description logics via alternating tree-automata. *Inf. Comput.*, 237:12–55, 2014.
17. J. Corman, J. L. Reutter, and O. Savkovic. Semantics and validation of recursive SHACL. In *International Semantic Web Conference*, pages 318–336, 2018.
18. S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF mapping language. W3C Recomm., 2011.
19. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, pages 89–124, 2005.
20. P. Heyvaert, B. De Meester, A. Dimou, and R. Verborgh. Declarative rules for Linked Data generation at your fingertips! In *The Semantic Web: ESWC Satellite Events*, pages 213–217, June 2018.
21. T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, September 1984.
22. H. Knublauch and D. Kontokostas. Shapes constraint language (SHACL). W3C Recomm., 2017.
23. J. E. Labra Gayo, E. Prud'hommeaux, I. Boneva, and D. Kontokostas. Validating RDF data. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 2017.
24. L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.
25. F. Michel, J. Montagnat, and C. Faron Zucker. A survey of RDB to RDF translation approaches and tools. Technical report, University Sophia Antipolis, 2013.

26. J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Semant.*, 8(4):255–270, 2010.
27. E. Prud'hommeaux, I. Boneva, J. Labra Gayo Emilio, and G. Kellogg. Shape expressions language 2.1. W3C Draft, 2018.
28. S. Staworko, I. Boneva, J. E. Labra Gayo, S. Hym, E. G. Prud'hommeaux, and H. R. Solbrig. Complexity and expressiveness of ShEx for RDF. In *International Conference on Database Theory (ICDT)*, pages 195–211, 2015.
29. S. Staworko and P. Wieczorek. Containment of shape expression schemas for RDF. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 303–319, 2019.
30. Y. Tzitzikas, C. Lantzaki, and D. Zeginis. Blank node matching and RDF/S comparison functions. In *International Semantic Web Conference (ISWC)*, pages 591–607, 2012.
31. B. Villazón and M. Hausenblas. R2RML and direct mapping test cases. W3C, 2012.
32. W3C. RDF validation workshop report: Practical assurances for quality RDF data, 2013.