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

Iovka Boneva, Lawek Staworko, Jose Lozano

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

Iovka Boneva, Sławek Staworko, and Jose Lozano

CRIStAL–UMR 9189, University of Lille, CNRS, and INRIA LINKS,
F-59000-Lille, France
{iovka.boneva, slawomir.staworko,
jose-martin.lozano-aparicio}@univ-lille.fr

**Abstract.** We investigate the data exchange from relational databases to RDF graphs inspired by R2RML with the addition of target shape schemas. 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). Finally, we present a number of results that show that relaxing the restrictions of the proposed framework leads to an increase in complexity.

## 1 Introduction

The recent decade has seen RDF raise to the task of interchanging data between Web applications [23]. 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 [16], Direct Mapping [4] or YARRRML [18]. As an example, consider the following R2RML mapping, itself an RDF 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 [31, 22]. One

of the benefits of working with data conforming to a schema is an increased execution safety: applications need not to worry about handling malformed or invalid data that could otherwise cause undesirable and difficult to predict side-effects. One family of proposed schema formalisms for RDF is based on *shape constraints* and this class includes *shape expressions schemas* (ShEx) [25, 26, 10] and *shape constraint language* (SHACL) [21, 14]. 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 has the following ShEx definition

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

Essentially, every employee IRI must have a single :name property, an optional :email property, and at least one :works property each leading to an IRI satisfying 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 *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. The above tgd is *full* i.e., it does not use existential quantifiers. 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 focus on full constructive st-tgds used with a set of non-overlapping IRI constructors and call this 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 [30]: 9 test cases use pattern-based function to transform data values and 7 test cases 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 restrict our attention to class of *deterministic shape schemas* that are at the intersection of ShEx and SHACL, are known to have desirable computational properties while remaining practical, and posses a sought-after feature of having an equivalent graphical representation (in the form of shape graphs) [27].

For a given consistent 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: a data exchange setting, consisting from the source schema, the set of st-tgds, and the target schema is *consistent* iff every consistent source database instance admits a solution. Because many solutions may be possible, the standard *possible word semantics* [19, 1] is applied when evaluating queries: a *certain answer* to a query over the target schema is an answer returned in every solution. Consequently, one is inclined 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 [17]. 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 [24].

The list of contributions of the present paper 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 deterministic 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 show that allowing nondeterministic target schemas makes the consistency problem $\Pi_2^p$-hard. We also present a generalization of our consistency characterization to include st-tgds with existential quantifiers but the extension is no longer in coNP and the lower bound remains an open question.
- We propose a novel notion of *universal simulation solution* that can be constructed for any consistent constructive relational to RDF data exchange setting. It allows to easily compute certain answers to any query class that is robust under graph simulation. We also apply existing results on relational to relational data exchange setting to show tractability of computing certain answers to conjunctive queries.
- We use the universal simulation solution to show tractability of computing certain answers to *forward nested regular expressions*. For the full class of *nested relational expressions* (NREs), considered to be the navigational core of SPARQL [24], we show an increase of complexity when computing certain answers.

In [11] we have studied the consistency problem for a more restrictive fully-typed data exchange setting, where all constructed IRIs must be typed. This restriction allowed to reduce the consistency problem to a simple test of functional dependencies propagation over relational views. This technique can no longer be employed for constructive data exchange setting, where the constructed RDF

nodes need not be typed, and to address it we propose a novel and non-trivial technique. In [11], we have also not considered certain query answering.

**Organization** The paper is organized as follows. In Section 2 we introduce the constructive data exchange framework with an illustrative example. In Section 3 we recall basic notions of relational and graph databases. In Section 4 we formalize the relational to RDF data exchange with IRI constructors and target shape schema. In Section 5 we study the problem of consistency. In Section 6 we study certain query answering. Section 7 contains a discussion of related work. And in Section 8 we present conclusions and outline future work.

# 2 Introductory Example

We illustrate the relational to RDF framework with the following example. We work with a *relational database* of software bug reports, presented in Figure 1. Each bug is reported by a user and a bug may have a number of related bugs. Each user may track a number of bugs.

*User*

| *uid* | *name* |
|---|---|
| 1 | Jose |
| 2 | Edith |

*Email*

| *uid* | *email* |
|---|---|
| 1 | j@ex.com |

*Track*

| *uid* | *bid* |
|---|---|
| 1 | 1 |
| 1 | 2 |

*Bug*

| *bid* | *descr* | *uid* |
|---|---|---|
| 1 | Boom! | 1 |
| 2 | Kabang! | 1 |
| 3 | Bang! | 2 |

*Rel*

| *bid* | *rid* |
|---|---|
| 2 | 1 |
| 1 | 3 |

Figure 1: Relational source database

We wish to export the contents of the above relational database to RDF for use by an existing application. The application expects the RDF document to adhere to the following ShEx schema (with `:` being the default prefix).

```
:TBug   { :descr xsd:string; :rep @:TUser; :rel @TBug * }
:TUser  { :name xsd:string; :email xsd:string; :tracks @TBug + }
```

This schema defines two types of nodes: TBug for bug reports and TUser for user info. This ShEx schema happens to closely mimic the structure of the relational database with two exceptions: the type TUser requires that every user must track at least one bug and must have a single email while the relational database is free of such constraints.

To assign an IRI to every user and every bug, we define two IRI constructors using the intuitive syntax of subject patterns of R2RML (where `bug:` and `usr:` are two IRI prefixes):

$$\mathit{bug2iri}(\mathit{bid}) = \texttt{"bug:}\{\mathit{bid}\}\texttt{"} \qquad \mathit{usr2iri}(\mathit{uid}) = \texttt{"usr:}\{\mathit{bid}\}\texttt{"}$$

Now, the R2RML mapping is formalized using the following set full constructive

dependencies.

$$
\begin{aligned}
\mathit{Bug}(b,d,u) &\Rightarrow \mathit{Triple}(\mathit{bug2iri}(b), \mathsf{:descr}, d) \wedge \mathsf{TBug}(\mathit{bug2iri}(b)) \wedge \\
&\quad\ \mathit{Triple}(\mathit{bug2iri}(b), \mathsf{:rep}, \mathit{usr2iri}(u)) \\
\mathit{Rel}(b_1,b_2) &\Rightarrow \mathit{Triple}(\mathit{bug2iri}(b_1), \mathsf{:related}, \mathit{bug2iri}(b_2)) \\
\mathit{User}(u,n) &\Rightarrow \mathit{Triple}(\mathit{pers2iri}(u), \mathsf{:name}, n) \\
\mathit{User}(u,n) \wedge \mathit{Track}(u,b) &\Rightarrow \mathit{Triple}(\mathit{usr2iri}(u), \mathsf{:tracks}, \mathit{bug2iri}(b)) \\
\mathit{User}(u,n) \wedge \mathit{Email}(u,e) &\Rightarrow \mathit{Triple}(\mathit{usr2iri}(u), \mathsf{:email}, e)
\end{aligned}
$$

One possible solution to the task at hand is presented in Figure 2. We point out

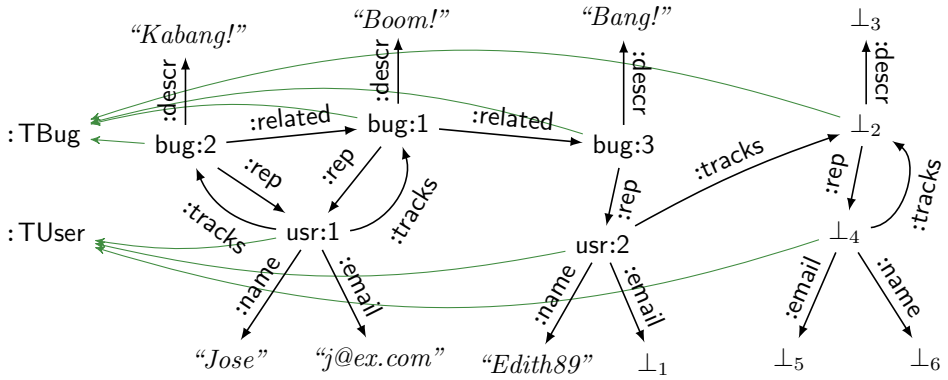


Figure 2: Target RDF graph (solution). Green thin arrows indicate types of non-literal nodes.

that a number of null values, for both IRI and literal nodes, has been introduced in the solution to make sure it satisfies the shape schema.

# 3 Preliminaries

In this section we recall basic notions of relational and graph databases. More formal definitions can be found in appendix.

**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 $\Sigma_{\text{fd}}$ 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 $\mathsf{Lit}$ of constants (also called literal values). The instance $I$ is *consistent* if it satisfies all functional dependencies $\Sigma_{\text{fd}}$.

**Graphs** An RDF graph $G$ is an labeled graph whose nodes are divided into two *kinds*: *literal* nodes and *non-literal* nodes with only non-literal nodes allowed to have outgoing edges. Every node is labeled but the label might be a *named null*. The type of value used depends on the kind of a node: literal nodes are labeled with *literal values* $\mathsf{Lit}$ and *literal null values* $\mathsf{NullLit}$ while non-literal node can be labeled with resource names $\mathsf{Iri}$ and null resource names $\mathsf{NullIri}$. More importantly, we adopt the *unique name assumption* (UNA) i.e., no two node

have the same label, and 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 $\mathsf{Pred} \subset \mathsf{Iri}$. We often view a graph as a set of *subject-predicate-object triples*.

**Shape Schemas** A *shapes schema* is a pair $\mathbf{S} = (\mathscr{T}, \delta)$, where $\mathscr{T}$ is a finite set of *type names* and $\delta \subseteq \mathscr{T} \times \mathsf{Pred} \times (\mathscr{T} \cup \{Literal\}) \times \{\mathtt{1}, \mathtt{?}, \mathtt{*}, \mathtt{+}\}$ is a set of shape constraints. A *shape constraint* $(T, p, S, \mu)$ 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 $\mu$: precisely one if $\mu = \mathtt{1}$, at most one if $\mu = \mathtt{?}$, at least one if $\mu = \mathtt{+}$, and arbitrarily many if $\mu = \mathtt{*}$. Naturally, the validity of a graph $G$ w.r.t. $\mathbf{S}$ is defined relative to a typing, a function $typing : nodes(G) \to \mathscr{T} \cup \{Literal\}$ that assigns to every non-literal node a set of types in $\mathscr{T}$ and to every literal node the special type label *Literal*. A *typed graph* $(G, typing)$ is *valid* w.r.t. $\mathbf{S}$ if every shape constraint of $\mathbf{S}$ is satisfied relative to *typing*.

We work only with *deterministic* shape schemas such that for every type $T \in \mathscr{T}$ and every predicate $p \in \mathsf{Pred}$ there is at most one shape constraints with $T$ and $p$. Consequently, we view $\delta$ as a partial function $\delta : \mathscr{T} \times \mathsf{Pred} \to (\mathscr{T} \cup \{Literal\}) \times \{\mathtt{1}, \mathtt{?}, \mathtt{*}, \mathtt{+}\}$ and set $\delta(T, p) = S^{\mu}$ whenever $(T, p, S, \mu) \in \delta$. We point out that deterministic shape schemas are expressible in both ShEx and SHACL.

**Dependencies** We employ the standard syntax of first-order logic and given a relational schema $\mathbf{R}$ and a shape schema $\mathbf{S}$, the vocabulary used to construct formulas comprises of the relation names of $\mathbf{R}$, a ternary predicate *Triple* for defining graph topology, and the types of $\mathbf{S}$ used as monadic predicates. We also the edge labels $\mathsf{Pred}$ as constant symbols with their straightforward interpretation. Naturally, we use of the equality relation $=$ and but by *clause* we understand a conjunction of (positive) atomic formulas that does not use $=$. Later on, we additionally introduce functions that allow to map the values in relational databases to resource names used in RDF graphs, and we shall allow the use of their 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 $\varphi$ is called the *body* and $\psi$ the *head* of the dependency, and we typically omit the universally quantified variables and write simply $\varphi \Rightarrow \bar{y}.\psi$. A dependency is *equality-generating* (egd) if its body is a clause and its head consists of an equality condition $x = y$ on pairs 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. For instance, the key dependency $User : uid \to name$ in the example in Section 2 can be expressed as $User(x, y_1) \wedge User(x, y_2) \Rightarrow y_1 = y_2$. 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: $T(x) \wedge Triple(x, p, y) \Rightarrow S(y)$,

**(PF)** the *predicate functionality* rule: $T(x) \wedge Triple(x, p, y_1) \wedge Triple(x, p, y_2) \Rightarrow y_1 = y_2$

if $\mu = \mathtt{1}$ or $\mu = \mathtt{?}$,

**(PE)** the *predicate existence* rule: $T(x) \Rightarrow \exists y.\ \mathit{Triple}(x, p, y)$ if $\mu = \mathtt{1}$ or $\mu = \mathtt{+}$.

**Chase** We use the standard notion of *homomorphism* and its extensions to formulas and sets of facts (relational structures). The *chase* is a procedure used to construct a solution for data exchange, and it begins with the source instance and iteratively executes any dependencies that are triggered. More precisely, a dependency $\sigma = \varphi \Rightarrow \exists \bar{y}.\psi$ is *triggered* in instance $I$ by a homomorphism $h$ if $h(\varphi) \subseteq I$ and there is no extension $h'$ of $h$ with $h'(\psi) \subseteq I$. The *execution* of $\sigma$ triggered in $I$ by $h$ may result in 1) adding new facts to $I$ when $\sigma$ is a tgd, 2) in renaming named null in $I$ when $\sigma$ is an egd, or 3) in a *failure* if $\sigma$ is an egd and $\psi$ contains a value equality $x = y$ but $h(x)$ and $h(y)$ are two different constants.

# 4 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* $\Sigma_{\mathrm{st}}$ *is a set of* source-to-target tuple generating dependencies (*st-tgds*) *whose bodies are formulas over* $\mathscr{R}$ *and heads are formulas over* $\mathscr{F} \cup \mathscr{T} \cup \{\mathit{Literal}\}$*.* $\mathscr{E}$ *is* constructive *if the library of IRI constructors is non-overlapping and the st-tgds* $\Sigma_{\mathrm{st}}$ *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* $I \cup J \cup F \models \Sigma_{\mathrm{st}}$*. By* $\mathit{sol}_{\mathscr{E}}(I)$ *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 *core pre-solution* for $I$ to $\mathscr{E}$ is the result $J_0$ of chase on $I$ with the st-tgds $\Sigma_{\mathrm{st}}$ and all **TP** rules of $\mathbf{S}$. In essence $J_0$ isobtained by exporting the relational data to RDF triples with $\Sigma_{\mathrm{st}}$ 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 $J \in \mathit{sol}_{\mathscr{E}}(I)$.

# 5 Consistency

In this section we study the problem of consistency of data exchange settings. The following notion of consistency was called absolute consistency in [8].

**Definition 2 (Consistency)** *A data exchange setting $\mathscr{E}$ is* consistent *if every consistent source instance $I$ of $\mathbf{R}$ admits a solution to $\mathscr{E}$.*

First we show that a constructive data exchange setting $\mathscr{E}$ is consistent if and only if it is value consistent (see Section 5.1) and node kind consistent (see Section 5.2) and the decision procedure is co-NP complete. Then in Section 5.3 we show that consistency checking is more complex for two more general data exchange settings.

## 5.1 Value Consistency

Value inconsistency captures situations in which all chase sequences would fail due to triggering a predicate functionality egd $T(x) \wedge \mathit{Triple}(x,p,y) \wedge \mathit{Triple}(x,p,y') \Rightarrow y = y'$ with a homomorphism that associates different constants with $y$ and $y'$. Let $\Sigma_{\mathbf{S}}^{\mathsf{TP}}$ be the set of type propagation rules and $\Sigma_{\mathbf{S}}^{\mathsf{PF}}$ be the set of predicate functionality rules from $\Sigma_{\mathbf{S}}$ as defined in Section 4.

**Definition 3 (Value consistent)** *Let $J$ be the core pre-solution for some source instance $I$ to $\mathscr{E}$. $J$ is* value consistent *if $J \models \Sigma_{\mathbf{S}}^{\mathsf{PF}}$. The data exchange setting $\mathscr{E}$ is* value consistent *if for every $I$ instance of $\mathbf{R}$, the core pre-solution for $I$ to $\mathscr{E}$ is value consistent.*

We now concentrate on identifying whether core pre-solutions to $\mathscr{E}$ satisfy $\Sigma_{\mathbf{S}}^{\mathsf{PF}}$. A triple of facts $W = \{T(f(\bar{a})), \mathit{Triple}(f(\bar{a}),p,b), \mathit{Triple}(f(\bar{a}),p,b')\}$ is called a *violation* if the definition of type $T$ contains a triple constraint of the form $p :: S^{1}$ or $p :: S^{?}$, and $b \neq b'$ are constants. The triple $(T,f,p)$ is called the *sort* of the violation.

We fix a violation $W = \{T(f(\bar{a})), \mathit{Triple}(f(\bar{a}),p,b), \mathit{Triple}(f(\bar{a}),p,b')\}$ for the sequel, and we explain how to check whether the dependencies in $\mathscr{E}$ allow to generate this violation. The proof goes by constructing a finite set $V$ of source instances s.t. $\mathscr{E}$ is value inconsistent iff there is an instance $I$ in $V$ s.t. chasing $I$ with $\Sigma_{\mathrm{fd}}$ fails. We start by an example illustrating some elements of the decision procedure.

**Example 4** *Let $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\mathrm{st}}, \mathbf{F})$ where $\mathbf{R} = (\mathscr{R}, \Sigma_{\mathrm{fd}})$, $\mathbf{F} = (\mathscr{F}, F)$, $\mathscr{R} = R, S$ both of arity two, and $\mathscr{F} = g_0, g, f$ all of arity one. The shapes schema is given by $\delta(U_0, r) = U^{\star}$, $\delta(U, q) = T^{\star}$, $\delta(T, p) = \mathit{Literal}^{1}\}$, and the st-tgds are as follows:*

(1) $R(x_0,x_1) \Rightarrow U_0(g_0(x_1))$
(2) $R(x_1,x_2) \Rightarrow \mathit{Triple}(g_0(x_1), r, g(x_2))$
(3) $R(x_2,x) \Rightarrow \mathit{Triple}(g(x_2), q, f(x))$
(4) $S(x,y) \Rightarrow \mathit{Triple}(f(x), p, y)$
(5) $R(x,z) \wedge S(x,y') \Rightarrow \mathit{Triple}(f(x'), p, y')$

*We want to construct a source instance s.t. when chased with $\mathscr{E}$ would produce a violation of sort $(T,p,f)$. First we need to produce a fact $T(f(x))$ for some $x$. This can be done by applying rules (1)–(3), then the type propagation rules for $\delta(U_0,r) = U^{\star}$ and $\delta(U,q) = T^{\star}$. More precisely, let $I_{123}$ be the instance obtained as the union of the bodies of rules (1)–(3) (where variables are used as elements of the domain). Note that the variables repeated between rules were chosen in such a way on purpose. The result of chasing $I_{123}$ by the above mentioned rules is $I' = I_{123} \cup \{U_0(g_0(x_1)), \mathit{Triple}(g_0(x_1), r, g(x_2)), U(g(x_2)), \mathit{Triple}(g(x_2), q, f(x)), T(f(x))\}$.*

*Now we want to use rules (4),(5) to obtain the two missing facts for the violation. For that, let $I_{123,4,5}$ be the union of $I_{123}$ and the bodies of rules (4),(5). Chasing $I_{123,4,5}$ with $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ we get its core pre-solution to $\mathscr{E}$: $J = I' \cup \{\mathit{Triple}(f(x), p, y), \mathit{Triple}(f(x'), p, y'), \mathit{Literal}(y), \mathit{Literal}(y')\}$ that contains a violation of sort $(T, p, f)$.*

*So far we didn't give the source dependencies on purpose. Suppose that the first attribute of $S$ is a primary key. In this case, $I_{123,4,5}$ is not a consistent source instance, and we can actually show that $\mathscr{E}$ is consistent. Without source dependencies, $\mathscr{E}$ is inconsistent, as witnessed by the source instance $I_{123,4,5}$.* □

Now we identify a necessary and sufficient condition for whether fact $T(f(\bar{a}))$ can appear in the pre-solutions to $\mathscr{E}$.

**Definition 5** *The pair $(T, f) \in \mathscr{T} \times \mathscr{F}$ is called* accessible in $\mathscr{E}$ with sequence $\sigma_0, \sigma_1, \ldots, \sigma_n$ *of st-tgds in $\Sigma_{\text{st}}$ if:*

- *the head of $\sigma_0$ is of the form $T_0(f_0(\bar{y}_0))$, and*
- *the head of $\sigma_i$ is of the form $\mathit{Triple}(f_{i-1}(\bar{x}_i), p_i, f_i(\bar{y}_i))$ for every $1 \leq i \leq n$, and*
- *$\delta(T_{i-1}, p_i) = T_i^{\mu_i}$ for every $0 \leq i < n$, and*
- *$T = T_n$ and $f = f_n$.*

*for some type symbols $T_i$, function symbols $f_i$, predicates $p_i$ and sequences of variables $\bar{x}_i$ and $\bar{y}_i$.*

Note that if $(T, f)$ is accessible in $\mathscr{E}$, then it is accessible with an elementary sequence $\sigma_0, \ldots, \sigma_n$ which elements are pairwise distinct.

In Example 4, $(T, f)$ is accessible in $\mathscr{E}$ with sequence (1)(2)(3).

The pairs $(T', f')$ accessible in $\mathscr{E}$ characterize the type facts that appear in the core pre-solutions to $\mathscr{E}$, as follows.

**Lemma 6** *For any $(T, f) \in \mathscr{T} \times \mathscr{F}$ it holds: $(T, f)$ is accessible in $\mathscr{E}$ if and only if there exists an instance $I$ of $\mathscr{R}$ and a tuple of constants $\bar{a}$ in the domain of $I$ s.t. the core pre-solution for $I$ to $\mathscr{E}$ contains the fact $T(f(\bar{a}))$.*

*Proof.* [Sketch of proof.] For the left-to-right direction, let $b \in \mathsf{Dom}$. Consider the instance $I$ that contains exactly one fact $R(b^{arity(R)})$ for any relational symbol $R$ in $\mathscr{R}$, where $b^n$ is the $n$-tuple containing only $b$'s. Then we show that $J_0$, the core pre-solution for $I$ to $\mathscr{E}$, contains the fact $T(f^F(b^{arity(f)}))$ whenever $(T, f)$ is accessible in $\mathscr{E}$.

For the right-to-left direction, we fix an arbitrary terminating chase sequence $s$ on $I$ with $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ producing the core pre-solution $J_0$. Using that $T(a)$ is a fact in $J_0$, we show that $s$ necessarily contains chase steps with dependencies as those Definition 5, thus witnessing that $(T, f)$ is accessible in $\mathscr{E}$.

Now we assume that the fact $T(f(\bar{a}))$ appears in the core pre-solutions for $I$ to $\mathscr{E}$ and want to verify whether the facts $\mathit{Triple}(f(\bar{a}), p, b)$, $\mathit{Triple}(f(\bar{a}), p, b')$ co-occur with it. Recall that $b, b'$ are constants, so such facts are necessarily generated by st-tgds. Two st-tgds $\sigma, \sigma'$ are called *contentious* with sort $(T, p, f)$ if the head of $\sigma$ is $\mathit{Triple}(f(\bar{z}), p, t)$, the head of $\sigma'$ is $\mathit{Triple}(f(\bar{z}'), p, t')$ and $(T, f)$ is accessible in $\mathscr{E}$, and predicate p is functional for type $T$, i.e. $\delta(T, p) = S^{\mu}$

with $\mu$ equal to 1 or ?. Note that $\sigma, \sigma'$ may be the same st-tgd, in which case we consider that they are two copies of it obtained by alpha renaming.

Suppose now that $\sigma, \sigma'$ are the contentious st-tgds here above, and that $\pi = \sigma_0, \ldots, \sigma_n$ is a sequence of st-tgds s.t. $(T, f)$ is accessible in $\mathscr{E}$ with $\pi$. We define a source instance $I_{\pi,\sigma,\sigma'}$ such that a chase sequence with rules $\sigma_0, \ldots, \sigma_n, \sigma, \sigma'$ can be executed on $I_{\pi,\sigma,\sigma'}$ yielding an instance that includes the violation $W$. Let $\sigma_{n+1} = \sigma$ and $\sigma_{n+2} = \sigma'$. Suppose w.l.o.g. that $\sigma_i$ and $\sigma_j$ use mutually disjoint sets of variables whenever $i \neq j$. Define $B_{\pi,\sigma,\sigma'} = \bigcup_{i=0}^{n+2} \mathit{body}(\sigma_i)$ where $\mathit{body}(\sigma_i)$ is the body of $\sigma_i$. Let $\sigma_0, \ldots, \sigma_n$ be as in Definition 5, thus $(T_n, f_n, p_n) = (T, f, p)$. Define the sequence of mappings $h_0, \ldots, h_{n+2}$ inductively as follows:

- for any $0 \leq i \leq n+2$, $h_i : \bigcup_{j=0}^{i} \mathit{vars}(\sigma_j) \to \mathsf{NullLit}$ is a mapping that is injective when restricted on $\mathit{vars}(\sigma_i)$, where $\mathit{vars}(\sigma)$ denotes the set of variables that appear in $\sigma$.
- for any $1 \leq i \leq n+2$, $h_i$ coincides with $h_{i-1}$ on the domain of $h_{i-1}$;
- for any $1 \leq i \leq n$, $h_i(\bar{x}_i) = h_{i-1}(\bar{y}_{i-1})$ and $h_i(z)$ is fresh w.r.t. the image of $h_{n-1}$ for any $z \notin \bar{x}_i$. That is, $z \notin \bar{x}_i$ implies $h(z)$ is not in the image of $h_{i-1}$;
- $h_{n+1}(\bar{z}) = h_n(\bar{y}_n)$ and $h_{n+1}(z)$ is fresh w.r.t. the image of $h_n$ for any $z \notin \bar{z}$;
- $h_{n+2}(\bar{z}') = h_n(\bar{y}_n)$ and $h_{n+2}(z)$ is fresh w.r.t. the image of $h_{n+1}$ for any $z \notin \bar{z}'$,

Then we let $h_{\pi,\sigma,\sigma'} = h_{n+2}$ and $I_{\pi,\sigma,\sigma'} = h_{\pi,\sigma,\sigma'}(B_{\pi,\sigma,\sigma'})$. It immediately follows from the definition that $h_{\pi,\sigma,\sigma'} : B_{\pi,\sigma,\sigma'} \to I_{\pi,\sigma,\sigma'}$ is a homomorphism. Moreover, it is easy to see that $I_{\pi,\sigma,\sigma'}$ is unique up to isomorphism, so from now on by $I_{\pi,\sigma,\sigma'}$ we mean an arbitrary instance isomorphic to the one defined above. In Example 4, the instance $I_{123,4,5}$ was obtained as described above.

The following proposition establishes an equivalence between the presence of the violation $W$ in the core pre-solution of an instance $I$ of $\mathscr{R}$, and the existence of a homomorphism from some $I_{\pi,\sigma,\sigma'}$ to $I$.

**Proposition 7** *Let $I$ be an instance of $\mathscr{R}$.*

1. *There exist $\pi, \sigma, \sigma, h$ s.t. $(T, f)$ is accessible in $\mathscr{E}$ with path $\pi$, $\sigma, \sigma'$ are contentious st-tgds of sort $(T, f, p)$, and $h : I_{\pi,\sigma,\sigma'} \to I$ is a homomorphism if and only if there exist a tuple of constants $\bar{a}$ from the domain of $I$ and constants $b, b'$ s.t. the core pre-solution for $I$ to $\mathscr{E}$ includes $\{T(f(\bar{a})), \mathit{Triple}(f(\bar{a}), p, b), \mathit{Triple}(f(\bar{a}), p, b')\}$.*
2. *Moreover, if the head of $\sigma$ is $\mathit{Triple}(f(\bar{z}), p, t)$ and the head of $\sigma'$ is $\mathit{Triple}(f(\bar{z}'), p, t')$, then $\bar{a} = h \circ h_{\pi,\sigma,\sigma'}(\bar{z}) = h \circ h_{\pi,\sigma,\sigma'}(\bar{z}')$, $b = h \circ h_{\pi,\sigma,\sigma'}(t)$ and $b' = h \circ h_{\pi,\sigma,\sigma'}(t')$.*

We point out that Proposition 7 identifies a necessary condition for the presence of some violation in the core pre-solution for a source instance $I$. The condition is not sufficient for two reasons. First, $I$ is an instance of $\mathscr{R}$ that does not necessarily satisfy the source functional dependencies. Second, $b$ might be equal to $b'$. Theorem 8 adds sufficient conditions for handling these two missing cases.

**Theorem 8** *These two statements are equivalent:*

- *For every instance $I$ of $\mathbf{R}$, the core pre-solution for $I$ to $\mathscr{E}$ is value consistent.*
- *For every violation sort $(T, f, p)$, every $\pi$ s.t. $(T, f)$ is accessible in $\mathscr{E}$ with $\pi$, any two contentious st-tgds $\sigma, \sigma'$ of sort $(T, f, p)$, every $J$ solution for $I_{\pi,\sigma,\sigma'}$ to $\Sigma_{\text{fd}}$ it holds that $(h_{\pi,\sigma,\sigma'} \circ h)(t) = (h_{\pi,\sigma,\sigma'} \circ h)(t')$, where $t, t'$ are such that the head of $\sigma$ is $Triple(f(\bar{z}), p, t)$ and the head of $\sigma'$ is $Triple(f(\bar{z'}), p, t')$, and $h$ is the unique homomorphism from $I_{\pi,\sigma,\sigma'}$ to $J$.*

## 5.2 Node Kind Consistency

Node kind inconsistency characterizes situations in which all chase sequences would fail due to the necessity of equating a literal and a non literal value by triggering a predicate functionality egd $T(x) \wedge Triple(x, p, y) \wedge Triple(x, p, y') \Rightarrow y = y'$ with homomorphism $h$ s.t. *exactly one* among $h(y), h(y')$ is a literal. In this case the corresponding chase sequence fails even if one of $h(y), h(y')$ is null. This is a particularity of relational to RDF data exchange (in contrast to relational data exchange).

In the sequel we give a definition of node kind consistency and announce the propositions needed for proving the consistency theorem. The detailed definitions are rather technical and are presented in Appendix A.1.

For a typed graph $J$ we define the set $CoTypes(J)$ of sets of *types co-occurring* in all solutions $G$ of $J$ to $\mathscr{E}$ that include $J$. That is, $X \in CoTypes(J)$ if for any $G$ s.t. $J \subseteq G$ and $G \in sol_{\mathscr{E}}(J)$, there exists a node $n$ in $G$ s.t. $X = \{T \in \mathscr{T} \cup \{Literal\} \mid T(n) \in G\}$.

**Definition 9 (Node kind consistent)** *Let $I$ be a source instance and $J$ its core pre-solution to $\mathscr{E}$. $J$ is* node kind consistent *if $CoTypes(J)$ does not contain a set $X$ s.t. $\{Literal, T\} \subseteq X$ for some type $T$ in $\mathscr{T}$. The data exchange setting $\mathscr{E}$ is node kind consistent if for every $I$ instance of $\mathbf{R}$, the core pre-solution for $I$ to $\mathscr{E}$ is value consistent.*

Node kind inconsistency is a sufficient condition for inconsistency.

**Lemma 10** *For any $I$ instance of $\mathbf{R}$, if the core pre-solution for $I$ to $\mathscr{E}$ is value inconsistent, then $I$ does not admit a solution to $\mathscr{E}$.*

In Theorem 8 we have shown that value inconsistency is another such sufficient condition. The next lemma establishes that being value consistent and node kind consistent is a sufficient condition for $\mathscr{E}$ to be consistent.

**Lemma 11** *For any $I$ instance of $\mathbf{R}$, if the core pre-solution for $I$ to $\mathscr{E}$ is value consistent and node kind consistent, then $I$ admits a solution to $\mathscr{E}$.*

We are now ready to establish our main results regarding consistency of constructive data exchange settings. The next theorem follows from Theorem 8, Lemma 10, Lemma 11, and the fact that value consistency and node kind consistency are decidable.

**Theorem 12 (Consistency)** *$\mathscr{E}$ is consistent iff $\mathscr{E}$ is value consistent and node kind consistent.*

Finally, we show that checking consistency of a constructive data exchange setting is co-NP complete. The lower bound is shown using a reduction to the complement of SAT.

**Theorem 13 (Complexity of consistency)** *Checking consistency of a constructive relational to RDF data exchange setting is coNP-complete.*

## 5.3 Non-Constructive st-tgds, Non-Deterministic Shape Schemas

The consistency checking algorithm can be extended to non-constructive data exchange settings but the lower co-NP complexity bound is not preserved by the extension. Consider a data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$ with $\mathbf{S} = (\mathscr{T}, \delta)$ in which the st-tgds in $\Sigma_{\text{st}}$ can contain existential rules of the form $\varphi \Rightarrow \exists \bar{y}.\psi$ where function terms use only universally quantified variables. We illustrate consistency checking on an example.

**Example 14** *Consider shapes schema with types $T, U$ and rules $\delta(T, p) = U^1$ and $\delta(U, q) = \mathit{Literal}^?$, and the st-tgds*

$$R(x, y, w) \Rightarrow \mathit{Triple}(f(x), p, g(y))$$
$$S(x', y') \Rightarrow \exists z'.T(f(x')) \wedge \mathit{Triple}(f(x'), p, z') \wedge \mathit{Triple}(z', q, y')$$
$$R(x'', y'', w'') \Rightarrow \mathit{Triple}(g(y''), q, w'')$$

*Even in presence of existential variables, we can statically infer that the st-tgd head atoms $\mathit{Triple}(z', q, y')$ and $\mathit{Triple}(g(y''), q, w'')$ are* contentious*, then construct the source instance $I = \{R(x, y, z), S(x, y'), R(x'', y, z'')\}$ witness of value inconsistency of the data exchange setting at hand. Indeed, the core pre-solution to $I$ contains the facts $\{\mathit{Triple}(f(x), p, g(y)), T(f(x)), \mathit{Triple}(f(x), p, \bot_1), \mathit{Triple}(\bot_1, q, y'), U(\bot_1), \mathit{Triple}(g(y), q, w$ Triggering the predicate functionality rule for $\delta(T, p)$ we equate $\bot_1$ with $g(y)$. Then the last three atoms in $I$ constitute a violation of the predicate functionality rule for $\delta(U, q)$.*

*The instance $I$ is discovered by exploring the possible interactions between the rules coming from the shape schema and the st-tgds' heads. We first remark that the terms $f(x)$ and $f(x')$ are* equatable *(i.e. the target values produced by them might be equal as they are produced by the same IRI constructor), then type $T$ is* accessible *for $f(x)$ due to the second st-tgd. The terms $g(y)$ and $z'$ are also equatable due to predicate functionality of $p$ for type $T$, and so are $g(y)$ and $g(y'')$ (same IRI constructor). Also, type $U$ is accessible for $g(y)$, so also for $z'$ and $g(y'')$ (type propagation of $\delta(T, p) = U^1$). Thus the target atoms (generated during chase from) $\mathit{Triple}(z', q, y')$ and $\mathit{Triple}(g(y''), q, w'')$ can both have as subject the same value $g(y)$, and trigger a violation due to the predicate functionality $\delta(U, q)$.* □

Similarly to the case of constructive data exchange settings, the consistency checking algorithm is based on the fact that $\mathscr{E}$ is value inconsistent iff there exists a value inconsistent instance $I$ among a finite set $V$ of source instances.

The latter is characterized by the presence of contentious atoms in st-tgds' heads, which in turn are discovered by a Datalog program. Formal definitions and description of the algorithm are given in Appendix A.2, and allow us to establish

**Theorem 15** *Consistency is decidable for data exchange settings with existential st-tgds.*

The exact complexity of the decision procedure is left for future work.

Finally we point out that if we consider non-deterministic shape schemas, then the complexity of checknig consistency increases. The proof is given in Appendix A.3.

**Theorem 16** *Checking consistency of a constructive relational to RDF data exchange setting with nondeterministic shape schema is* $\Pi_2^p$*-hard.*

# 6 Certain Query Answering

In this section we investigate computing certain query answers. We focus mainly on Boolean queries as it allows us to more easily present our constructions and compare various classes of queries; later on we extend our results to non-Boolean queries. Throughout this section we fix a constructive data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$ and assume $\mathscr{E}$ is consistent. We recall that for a Boolean graph query $Q$, *true* is the *certain answer* to a query $Q$ 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 if the query is non-Boolean, we drop any answers that use null values). However, in the case of consistent constructive relational to RDF data exchange, a finite universal solution may not exists as it is the case in the example in Section 2. Indeed, the mutually recursive types TBug and TUser cause the chase to loop ad infinitum: the user *Edith* results in the node `usr:2` of type TUser which required to track at least one problem. Since in the relational database instance *Edith* does not track any bug, the chase needs to "invent" a fresh null IRI of type TBug. This node is required to have a user that has reported it and again the chase "invents" another fresh null IRI of type TUser, and so on.

Instead, we construct a solution, where we avoid inventing nodes with the same set of types, thus creating loops as illustrated in Figure 2. While this solution is not universal, it seems quite natural, and interestingly, we show that 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 consistent answers under data complexity. We show that extending this fragment to full nested regular expression leads to significant complexity increase. Finally, we also show that existing result on chase with guarded tdgs and egds can be used to compute certain answers to conjunctive queries.

**Nested regular expressions** In this paper we focus mainly on the class of *nested regular expressions* (NREs) that have been proposed as the navigational core of SPARQL [24]. In essence, NREs are regular expressions that use concatenation $\cdot$, union $+$, Kleene's closure $*$, inverse $-$, and permit nesting and testing node and edge labels. Formally, NREs are defined with the following grammar:

$$E ::= \epsilon \mid p \mid \Box \mid \langle \ell \rangle \mid [E] \mid E^* \mid E^- \mid E \cdot E \mid E + E$$

where $p \in \mathsf{Pred}$, $\ell \in \mathsf{Iri} \cup \mathsf{Lit}$, and $\Box$ is a distinguished wildcard predicate symbol. An NRE is *forward* ($\text{NRE}^{\rightarrow}$) if it does not use the inverse operator. An NRE $E$ defines a binary relation $[\![E]\!]_G$ on nodes of a graph $G$ as follows.

$$\begin{array}{ll}
[\![\epsilon]\!]_G = \{(n,n) \mid n \in \mathit{nodes}(G)\}, & [\![[E]]\!]_G = \{(n,n) \mid \exists m.\, (n,m) \in [\![E]\!]_G\}, \\
[\![p]\!]_G = \{(n,m) \mid (n,p,m) \in G\}, & [\![E_1 + E_2]\!]_G = [\![E_1]\!]_G \cup [\![E_2]\!]_G, \\
[\![\Box]\!]_G = \{(n,m) \mid \exists p \in \mathsf{Iri}.\, (n,p,m) \in G\}, & [\![E_1 \cdot E_2]\!]_G = [\![E_1]\!]_G \circ [\![E_2]\!]_G, \\
[\![\langle \ell \rangle]\!]_G = \{(n,n) \mid n \in \mathit{nodes}(G) \wedge n = \ell\}, & [\![E^*]\!]_G = [\![E]\!]_G^*, \qquad [\![E^-]\!]_G = [\![E]\!]_G^{-1}.
\end{array}$$

An NRE $E$ is *satisfied* in a graph $G$ iff $[\![E]\!]_G \neq \emptyset$. We point out that NREs are incompatible with conjunctive queries but even forward NREs capture the subclass of acyclic conjunctive queries. Also, NREs (forward NREs) properly captures 2-way regular path queries (regular path queries, resp.)

## 6.1 Universal simulation solution

**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 simulation because it captures the essence of exploring a graph by means of following outgoing edges only.

**Definition 17** *A class $\mathscr{Q}$ of Boolean queries on graphs is* robust under simulation *iff for any query $Q \in \mathscr{Q}$ 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 $H$.* □

The class of patterns presented above has this very property, which is shown with an induction on the structure of the query. We point out, however, that our approach is not restricted to forward NREs only.

**Lemma 18** *The class of forward nested regular expressions is robust under simulation.*

The related notion of bisimulation has found application in normalizing blank nodes and essentially minimizing RDF graphs without altering its informational

contents [29]. Formally, a *bisimulation* of a graph $G$ is a simulation $R$ of $G$ by $G$ that is symmetric and reflexive. Again, there exists a maximal bisimulation of any graph $G$, which we denote by $\leftrightarrow\!\!\!\!\rightarrow$. We use the maximal bisimulation of a graph $G$ to construct it *reduct* $G/_{\leftrightarrow\!\!\!\!\rightarrow}$, which is the standard the quotient of the graph $G$ and the equivalence relation $\leftrightarrow\!\!\!\!\rightarrow$ and replaces nodes of every equivalence class by a single representative (details in appendix). The main property that we employ in our proofs is that of the reduct of a typed graph satisfies precisely the same shape schemas and the same queries from any class robust under simulation, and furthermore it is the smallest typed graph to have this property.

**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 allowing to find all certain answers.

**Definition 19** *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 20** *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 any universal simulation solution to $\mathscr{E}$ for $I$.*

The main challenge remains in constructing a universal simulation solution. We begin with the *core pre-solution* $J_0$ for $I$, which is the unique minimal typed graph $J_0$ that satisfies the st-tgds $\Sigma_{\text{st}}$ and the **TP** rules for $\mathbf{S}$ (cf. Section 3). The core pre-solution $J_0$ does not necessarily satisfy $\mathbf{S}$ as it may have *frontier* nodes whose type requires outgoing edges that are missing. To identify such nodes and add the necessary outgoing edges we first identify the types associated to a node in a typed RDF graph $types_G(n) = \{T \mid T(n) \in G\}$. Also, we say that a type $T$ *requires an outgoing $p$-edge* if $p :: S^\mu \in \delta(T)$ for some $\mu \in \{1, +\}$ and some type $S$, and by $Req(X)$ we denote the set of all IRIs that is required by any $T$ in $X$. Now, the frontier of $J_0$ is the following set

$$\mathbb{F} = \{(n, p) \mid n \in nodes(J_0),\ p \in Req(types_{J_0}(n)),\ \nexists m.\ Triple(n, p, m) \in J_0\}.$$

We also define a function that for a set of types $X$ satisfied at a node indicates the set of types $\Delta(X, p)$ that must hold at any node reachable by $p$-labeled edge

$$\Delta(X, p) = \{S \mid p :: S^\mu \in \delta(T) \text{ for some } T \in X \text{ and } \mu \in \{?, 1, +, *\}\}$$

Now, the set of additional null nodes that we add to $J_0$ is constructed in an iterative process (where we identify each node with the set of types it is to satisfy): $N = \bigcup_{i=0}^{\infty} N_i$, where $N_0 = \{\Delta(types(n), p) \mid (n, p) \in \mathbb{F}\}$ and $N_i = \{\Delta(X, p) \mid X \in N_{i-1},\ p \in Req(X)\}$ for $i \geq 1$. Note that we construct only subsets of the finite set of types $\mathscr{T}$, and therefore, this process eventually reaches a fix point. It may however be of size exponential in the size of the schema, and in fact, with an elaborate example using Chinese reminder theorem we can

show that it is in fact a tight bound of our construction. Now, the additional component of a universal simulation solution is the following graph

$$G_{\mathbf{S}} = \{(n, p, \Delta(types_{J_0}(n), p)) \mid (n, p) \in \mathbb{F}\} \cup \\ \{(X, p, \Delta(X, p)) \mid X \in N \wedge p \in Req(X)\} \cup \{T(X) \mid X \in N \wedge T \in X\}.$$

We point out that $J_0 \cup G_{\mathbf{S}}$ does in fact satisfy $\Sigma_{\text{st}}$ and $\mathbf{S}$ but it may not be the minimal universal simulation solution. However, it suffices to take the bisimulation quotient of $G_{\mathbf{S}}$ to ensure the minimality: the constructed universal simulation solution is $\mathscr{U}_0 = J_0 \cup G_{\mathbf{S}}/_{\leftrightarrow}$. We point out that because $J_0$ does not have any null nodes, $\mathscr{U}_0 = (J_0 \cup G_{\mathbf{S}})/_{\leftrightarrow}$.

**Theorem 21** *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 $\mathbf{S}$.*

## 6.2 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 [24], we get the following result.

**Theorem 22** *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.*

**Full nested regular expressions** Computing certain answers to the full class of NRE remains an open question. One could explore using 2-way alternating automata (2ATAs) for infinite trees corresponding to unraveling the universal simulation solution $\mathscr{U}_0$, a method that has been successfully applied to the closely related problem of computing certain answers to variants of regular path queries in the presence of ontologies [13, 20]. However, using 2ATAs comes with significant computational cost, and indeed, we show an increase in the complexity of computing certain answers to NREs as compared to forward NREs. This increase is detected when we fix the data exchange setting but consider both the query and the source instance to be part of the input, a complexity measure that is between data and combined complexity measures. Formally, for a class of Boolean graph queries $\mathscr{Q}$ and a data exchange setting $\mathscr{E}$ we define the decision problem $D^{\mathscr{Q}}_{\mathscr{E}} = \{(I, Q) \mid Q \in \mathscr{Q},\ \textit{true}$ is the certain answer to $Q$ in $I$ w.r.t. $\mathscr{E}\}$.

**Proposition 23** *For any constructive relational to RDF data exchange setting $\mathscr{E}$, $D^{NRE^{\rightarrow}}_{\mathscr{E}}$ is in PTIME and $D^{NRE}_{\mathscr{E}}$ is PSPACE-hard.*

**Conjunctive queries** The set of tgds in $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}$ is guarded and as such enables using existing results by Calì et al. [12] on tractability of certain answering for conjunctive queries. We recall that the classes of conjunctive queries and NREs are incomparable.

**Proposition 24** *The data complexity of computing certain answers to conjunctive queries w.r.t. constructive relational to RDF data exchange setting is in PTIME.*

**Non-Boolean queries** So far we have considered only Boolean queries and now we illustrate, on the example of binary forward NREs, that the universal simulation solution can be used to compute certain answers using the well-known method of evaluating the query over $\mathscr{U}_0$ and dropping any answers using null values. Formally, a pair of nodes $(n, m)$ is an *answer* to an NRE $E$ in a graph $G$ iff $(n, m) \in [\![E]\!]_G$. A pair $(n, m)$ is a *certain answer* to an NRE $E$ in $I$ w.r.t. $\mathscr{E}$ iff $(n, m)$ is an answer in every solution for $I$ to $\mathscr{E}$.

**Proposition 25** *Given a constructive data exchange setting $\mathscr{E}$, a source instance $I$, and a forward NRE $E$, a pair $(n, m)$ is a certain answer to $E$ in $I$ w.r.t. $\mathscr{E}$ if and only if $(n, m)$ is an answer to $E$ in a universal simulation solution $\mathscr{U}$ for $I$ w.r.t. $\mathscr{E}$ and neither $n$ nor $m$ are null.*

The above result can be generalized to any class of non-Boolean queries that is robust under simulation. However, attempting to present a precise definition of non-Boolean queries robust under simulation would exceed the space limits and we leave it for the full version of the paper.

# 7 Related Work

R2RML is a W3C standard language for defining custom relational to RDF mappings [16], other languages such as YARRRML [18] are compiled to R2RML. These languages do not impose constraints on the target, and consequently, the solution is always defined, unique, and trivially consistent which makes the problems of consistency and certain query answering irrelevant. In [9], Boneva et al. have studied relational to graph data exchange with st-tgds and the target constraints based on conjunctions of nested regular expressions. The framework is incomparable to the framework presented in this paper.

Viewing RDF as a ternary relation and expressing shape constraints with a set of target dependencies, thus reducing our framework to the standard relational data exchange [17], allows us to translate back existing results but only to a certain degree. Most notably, the work on chase with guarded tgds [12] allows us to show that computing certain answers to conjunctive queries in our framework is tractable (Proposition 24). In general, other works consider dependencies that are unsuitable to capture our mappings and shape schemas, focus on query classes that are not as well suited to query RDF as are NREs, or being very generic incur a much higher computational cost. For instance, data exchange with weakly acyclic tgds and edgs [17] is suitable for capturing only a restricted *weakly-recursive* shape schemas [11] that do not result in an infinite chase. While there exist works on data exchange that consider queries that go beyond conjunctive queries and add elements of transitive closure, such as XML tree patterns [8, 2] or Datalog fragments [3], they come at a price of high complexity. Also, while shape schemas are reminiscent of DTDs (or more closely of XML Schemas), XML is an ordered model and the source to target mappings in XML data exchange need to specify the relative order of elements or

a universal solution may fail to exists, and even if unordered XML is employed computing certain answers easily becomes intractable [5]. Finally, there is work on answering classes of regular path queries in description logics [13, 7] allows to easily capture our constructive data exchange settings and the considered classes of queries seem suitable for querying RDF but again they come with significant computational cost. However, we believe that the underlying use of 2-way alternating tree automata (2ATA) [15, 28] can be employed to computing certain answers to NREs in our framework, which we intend to pursue in our future work.

# 8 Conclusion and Future Work

We have presented a data exchange framework for exporting in a R2RML-like fashion a relational database to RDF with (non-overlapping) IRI constructors and target shape schema. We have studied the problems of consistency and have shown it to be coNP-complete using an intricate characterization. We have also studied computing certain answers to forward nested regular expressions and shown it be tractable using a novel construction of universal simulation solution. We have also shown that extending the framework in a number of natural directions generally leads to an increase of complexity.

Future research directions include a complete complexity analysis of relational to RDF data exchange with non-constructive st-tgds and nondeterministic and disjunctive shape schemas, and exploring using 2ATA for computing certain query answers to the full fragment of nested relational expressions.

# References


[1] Serge Abiteboul, Paris Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds, 1989.

[2] Shun'ichi Amano, Claire David, Leonid Libkin, and Filip Murlak. XML Schema Mappings: Data Exchange and Metadata Management. *J. ACM*, 61(2):12:1–12:48, 2014.

[3] Marcelo Arenas, Pablo Barceló, and Juan L. Reutter. Query languages for data exchange: Beyond unions of conjunctive queries. *Theory Comput. Syst.*, 49(2):489–564, 2011.

[4] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. A Direct Mapping of Relational Data to RDF, 2012. URL: `https://www.w3.org/TR/rdb-direct-mapping/`.

[5] Marcelo Arenas and Leonid Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2):7:1–7:72, 2008.

[6] Sören Auer, Lee Feigenbaum, Daniel Miranker, Angela Fogarolli, and Juan Sequeda. Use Cases and Requirements for Mapping Relational Databases to RDF, 2010. URL: `https://www.w3.org/TR/rdb2rdf-ucr/`.

[7] Meghyn Bienvenu, Magdalena Ortiz, and Mantas Simkus. Regular path queries in lightweight description logics: Complexity and algorithms. *J. Artif. Intell. Res.*, 53:315–374, 2015.

[8] Mikołaj Bojańczyk, Leszek A. Kołodziejczyk, and Filip Murlak. Solutions in XML data exchange. *J. Comput. Syst. Sci.*, 79(6):785–815, 2013.

[9] Iovka Boneva, Angela Bonifati, and Radu Ciucanu. Graph Data Exchange with Target Constraints. In *EDBT/ICDT Workshops - Querying Graph Structured Data (GraphQ)*, pages 171–176, 2015.

[10] Iovka Boneva, Jose Emilio Labra Gayo, and Eric G. Prud'hommeaux. Semantics and Validation of Shapes Schemas for RDF. In *ISWC2017*, 2017.

[11] Iovka Boneva, Jose Lozano, and Slawomir Staworko. Relational to RDF Data Exchange in Presence of a Shape Expression Schema. In *Proceedings of AMW*, 2018.

[12] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:57 – 83, 2012.

[13] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Answering regular path queries in expressive description logics via alternating tree-automata. *Information and Computation*, 237:12 – 55, 2014.

[14] Julien Corman, Juan L. Reutter, and Ognjen Savkovic. Semantics and validation of recursive SHACL. In *The Semantic Web - ISWC 2018*, pages 318–336, 2018.

[15] S. S. Cosmadakis, H Gaifman, P. C. Kanellakis, and M. Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In *ACM SIAM Symposium on Discrete Algorithms (SODA)*, pages 477–490, 1988.

[16] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF mapping language (W3C recomendation), 2011. URL: `http://www.w3.org/TR/r2rml/`.

[17] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, pages 89–124, 2005.

[18] Pieter Heyvaert, Ben De Meester, Anastasia Dimou, and Ruben Verborgh. Declarative rules for linked data generation at your fingertips! In *The Semantic Web: ESWC 2018 Satellite Events*, pages 213–217, 2018.

[19] Tomasz Imieliński and Witold Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, September 1984. URL: `http://doi.acm.org/10.1145/1634.1886`, `doi:10.1145/1634.1886`.

[20] Jean Christoph Jung, Carsten Lutz, Mauricio Martel, and Thomas Schneider. Querying the Unary Negation Fragment with Regular Path Expressions. In *21st International Conference on Database Theory (ICDT 2018)*, pages 15:1–15:18, 2018.

[21] Holger Knublauch and Dimitris Kontokostas. Shapes Constraint Language (SHACL), 2017. URL: `https://www.w3.org/TR/shacl/`.

[22] Jose Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, and Dimitris Kontokostas. Validating RDF data. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 2017.

[23] Franck Michel, Johan Montagnat, and Catherine Faron Zucker. A survey of RDB to RDF translation approaches and tools. Technical report, University Sophia Antipolis, 2013. URL: `https://hal.archives-ouvertes.fr/hal-00903568`.

[24] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. nSPARQL: A navigational language for RDF. *Web Semantics: Science, Services and Agents on the World Wide Web*, pages 255 – 270, 2010.

[25] Eric Prud'hommeaux, Iovka Boneva, Jose Emilio Labra Gayo, and Gregg Kellogg. Shape Expressions Language 2.1, 2018. URL: `http://shex.io/shex-semantics/index.html`.

[26] Slawek Staworko, Iovka Boneva, Jose Emilio Labra Gayo, Samuel Hym, Eric G. Prud'hommeaux, and Harold R. Solbrig. Complexity and Expressiveness of ShEx for RDF. In *ICDT*, pages 195–211, 2015.

[27] Sławek Staworko and Piotr Wieczorek. Containment of Shape Expression Schemas for RDF. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2019.

[28] S. Tasiran, R. Hojati, and R. K. Brayton. Language containment of non-deterministic *omega*-automata. In *Correct Hardware Design and Verification Methods*, pages 261–277, 1995.

[29] Yannis Tzitzikas, Christina Lantzaki, and Dimitris Zeginis. Blank Node Matching and RDF/S Comparison Functions. In *Proceedings of the 11th International Conference on The Semantic Web*, pages 591–607, 2012.

[30] Boris Villazón and Michael Hausenblas. R2RML and Direct Mapping Test Cases, 2012. URL: `https://www.w3.org/TR/rdb2rdf-test-cases/`.

[31] W3C. RDF validation workshop report: Practical assurances for quality RDF data. Technical report, W3C, 2013. URL: `http://www.w3.org/2012/12/rdf-val/report`.

# A Omitted Formalisms and Proofs

**Constants, nulls, and variables** We assume a fixed enumerable domain $\mathsf{Dom}$. For the purposes of this paper, we assume the domain to be partitioned into three infinite subsets $\mathsf{Dom} = \mathsf{Iri} \cup \mathsf{Lit} \cup \mathsf{NullIri}$ of *IRIs*, *literals*, and *blank* node identifiers respectively. We assume an infinite set of *null literals* $\mathsf{NullLit} \subseteq \mathsf{Lit}$ and identify the set of *null values* $\mathsf{Null} = \mathsf{NullLit} \cup \mathsf{NullIri}$. We refer to the remaining elements as *constants* $\mathsf{Const} = \mathsf{Dom} \setminus \mathsf{Null}$, and additionally, identify non-null literals $\mathsf{ConstLit} = \mathsf{Lit} \setminus \mathsf{NullLit} = \mathsf{Const} \cap \mathsf{Lit}$. Also, we fix an infinite set of (first-order) variables $\mathsf{Vars}$. In the sequel, we use $a$, $b$, and $c$ to range over elements of $\mathsf{Dom}$ and $\bar{a}$, $\bar{b}$, $\bar{c}$ to range over sequences of elements of $\mathsf{Dom}$. Similarly, we use $x$, $y$, and $z$ to range over variables and $\bar{x}$, $\bar{y}$, and $\bar{z}$ to range over sequences of variables.

**First-order logic** We recall basic notions of first-order logic. We assume a finite *signature*, which consists of a set of relation symbols $\mathscr{R}$ and a set of function symbols $\mathscr{F}$, each symbol having a fixed arity. We use first-order formulas with relation and function names from $\mathscr{R} \cup \mathscr{F} \cup \{=\}$, variables from $\mathsf{Vars}$, and constants from $\mathsf{Dom}$, however, we only employ *flat terms* that do not use nested applications of function symbols. An *atom* has the form $R(\bar{t})$, where $R \in \mathscr{R}$ and $\bar{t}$ is a sequence of terms. A *relational atom* does not use any function symbols. A *clause* is a conjunction of atoms and we often view it as a set of atoms. A formula is *closed* if it has no free variables. A formula is *ground* if it uses no variables whatsoever. A *fact* is a ground relational atom. A clause is *relational* if it employs only relational atoms.

A *structure* (or a *model*) $M$ (over the signature $\mathscr{R} \cup \mathscr{F}$) is a mapping that associates to every relation and function symbol $\xi \in \mathscr{R} \cup \mathscr{F}$ a corresponding relation or function $\xi^M$ on elements of $\mathsf{Dom}$ of appropriate arity. The semantics of a first-order logic formula $\varphi$ over a model $M$ is captured with the *entailment relation* $M \models \varphi$ defined in the standard fashion. The entailment relation is extended to a set of formulas: $M \models \Phi$ iff $M \models \varphi$ for every $\varphi \in \Phi$. Also, we often view a model over a signature consisting of relation symbols only as the set of all facts satisfied by the model.

**Dependencies** A *dependency* $\sigma$ is a closed first-order formula of the form $\forall \bar{x}.\varphi \Rightarrow \exists \bar{y}.\psi$, and we define $\mathit{body}(\sigma) = \varphi$, $\mathit{head}(\sigma) = \psi$, and $\mathit{vars}(\sigma) = \bar{x} \cup \bar{y}$. $\sigma$ is an *equality-generating dependency* (egd) if $\varphi$ is a clause and $\psi$ is a conjunction of equality conditions $x = y$ on pairs of variables. $\sigma$ is a *tuple-generating dependency* (tgd) if both $\varphi$ and $\psi$ are clauses. A tgd is *full* if it uses no existentially quantified variables $\bar{y}$. The use of the equality relation $=$ in an egd $\sigma$ implies a binary relation on the variables of $\sigma$, its (reflexive, symmetric, and transitive) closure gives an equivalence relation that identifies variables that need to have the same value, and by $\mathit{eq\text{-}class}(\sigma)$ we denote the set of equivalence classes to which this relation partitions the variables $\bar{x}$ (the variables $\bar{y}$ can be ignored).

**Relational databases** A *relational schema* is a pair $\mathbf{R} = (\mathscr{R}, \Sigma_{\mathrm{fd}})$ where $\mathscr{R}$ is a set of relation names, each with a fixed arity, and $\Sigma_{\mathrm{fd}}$ is a set of *functional dependencies* (fds) of the form $R : X \to Y$, where $R \in \mathscr{R}$ is a relation name of arity $n$, and $X, Y \subseteq \{1, \ldots, n\}$. An fd $R : X \to Y$ is a short for the egd $\forall \bar{x}, \bar{y}.\ R(\bar{x}) \wedge R(\bar{y}) \wedge \bigwedge_{i \in X}(x_i = y_i) \Rightarrow \bigwedge_{j \in Y}(x_j = y_j)$. An *instance* of $\mathbf{R}$ is a model $I$ over $\mathscr{R}$, and unless we state otherwise, in the sequel we work only with

instances that use literal constants from $\mathsf{ConstLit}$. The *active domain* $dom(I)$ of the instance $I$ is the set of elements of $\mathsf{Dom}$ used in $I$.

**Graphs** An *RDF graph* (or simply a *graph*) is a finite set $G$ of triples in $(\mathsf{Iri} \cup \mathsf{NullIri}) \times \mathsf{Pred} \times (\mathsf{Iri} \cup \mathsf{NullIri} \cup \mathsf{Lit})$. We view $G$ as an edge labeled graph by interpreting a triple $(s, p, o)$ as a $p$-labeled edge from the node $s$ to the node $o$. The set of *nodes* of $G$, denoted $nodes(G)$, is the set of elements that appear on first or third position of a triple in $G$.

**Shape constraints as dependencies** A deterministic shapes schema $\mathbf{S}$, as defined in Section 3, can be captured with a set $\Sigma_{\mathbf{S}}$ of egds and tgds. $\Sigma_{\mathbf{S}}$ contains for all type $T \in \mathscr{T}$ and all shape constraint $\delta(T, p) = S^{\mu}$

- $\mathbf{TP}(T, p, S) = \forall x, y.\ T(x) \wedge \mathit{Triple}(x, p, y) \Rightarrow S(y)$,
- $\mathbf{PF}(T, p) = \forall x.\ T(x) \Rightarrow \exists y.\ \mathit{Triple}(x, p, y)$, if $\mu \in \{\mathtt{1}, \mathtt{+}\}$,
- $\mathbf{PE}(T, p) = \forall x, y, z.\ T(x) \wedge \mathit{Triple}(x, p, y) \wedge \mathit{Triple}(x, p, z) \Rightarrow y = z$, if $\mu \in \{\mathtt{1}, \mathtt{?}\}$.

It is easy to see that a typed graph $(G, \mathit{typing})$ satisfies $\mathbf{S}$ if and only if $(G, \mathit{typing}) \models \Sigma_{\mathbf{S}}$.

**Homomorphisms and universal solutions** A substitution is a function $h : \mathsf{Dom} \cup \mathsf{Vars} \to \mathsf{Dom} \cup \mathsf{Vars}$ that is different from identity on a finite set $dom(h)$ of null values and variables, and furthermore, $h$ assigns a value in $\mathsf{Dom}$ to every element in $dom(h)$. We assume that the library of IRI constructors is known from the context, and extend substitutions to flat terms while applying interpretations of the IRI constructors: $h(f(t_1, \ldots, t_k)) = f^F(h(t_1), \ldots, h(t_k))$. We further extend homomorphisms, in a standard fashion, to atoms $h(R(t_1, \ldots, t_k)) = R(h(t_1), \ldots, h(t_k))$, and to sets of atoms $h(A) = \{h(\alpha) \mid \alpha \in A\}$. Recall that both instances and clauses can be viewed as set of atoms. Now, a *homomorphism* of $I_1$ in $I_2$ is a substitution $h$ such that $h(I_1) \subseteq I_2$. A homomorphism $h'$ *extends* a homomorphism $h$, written $h \subseteq h'$, if $dom(h) \subseteq dom(h')$ and $h'(x) = h(x)$ for all $x \in dom(h)$. A *universal* solution $U \in sol_{\mathscr{E}}(I)$ is a solution that subsumes all other solutions i.e., for any $J \in sol_{\mathscr{E}}(I)$ there is a homomorphism of $U$ in $J$.

**Chase** We recall the *chase* procedure for tgds and egds. Let $\sigma = \forall \bar{x}.\ \varphi \Rightarrow \exists \bar{y}.\ \psi$. If $\sigma$ is a tgd, we say that it is *triggered* in $I$ by $h$ if $dom(h) = \bar{x}$, $h(\varphi) \subseteq I$, and there is no extension $h'$ of $h$ such that $h'(\psi) \subseteq I$. It has a successful *execution* $h'$ yielding $I'$, in symbols $I \xrightarrow{\sigma,h} I'$, if $h'$ is an extension of $h$ such that $dom(h') = \bar{x} \cup \bar{y}$ and $I' = I \cup h'(\psi)$.

Next, suppose that $\sigma$ is an egd. It is *triggered* in $I$ by $h$ if $dom(h) = \bar{x}$, $h(\varphi) \subseteq I$, and there is $\bar{z} \in \mathit{eq\text{-}class}(\sigma)$ and $z_1, z_2 \in \bar{z}$ such that $h(z_1) \neq h(z_2)$. It has a successful *execution* $I'$ with $h'$, in symbols $I \xrightarrow{\sigma,h'} I'$, if $I' = h'(I)$ and $h'$ is a homomorphism such that $dom(h') = h(dom(h)) \cap \mathsf{Null}$ i.e., $h'$ assigns values to the null values used by $h$, and for any $\bar{z} \in \mathit{eq\text{-}class}(\sigma)$ and any $z_1, z_2 \in \bar{z}$ we have $h'(h(z_1)) = h'(h(z_2))$. If $\sigma$ is triggered in $I$ by $h$ but does not have a successful execution, we say that it *fails*, in symbols $I \xrightarrow{\sigma,h} \bot$.

Now, a *chase* sequence on $I_0$ with a set $\Sigma$ of tgds and egds is a possibly infinite sequence $I_0 \xrightarrow{\sigma_0,h_0} I_1 \xrightarrow{\sigma_1,h_1} I_2 \ldots$, where $\sigma_i \in \Sigma$ for all $i$. A *terminating* chase sequence ends with a failure or an instance that triggers no dependency in $\Sigma$. It is a classic result that a universal solution $U$ for $I$ to $\Sigma$ exists if and

only if there is a terminating chase sequence on $I$ with $\Sigma$ that ends with $U$ [17]. Naturally, this result extends to constructive data exchange settings with fixed IRI constructors.

## A.1 Proofs for Section 5 (Consistency)

We start by precising the definition of a chase step in order to take into account the possible conflict due to merging a literal and a non-literal node. Suppose that $\sigma$ is an egd. It is *triggered* in $I$ by $h$ if $dom(h) = \bar{x}$, $h(\varphi) \subseteq I$, and there is $\bar{z} \in$ *eq-class*$(\sigma)$ and $z_1, z_2 \in \bar{z}$ such that $h(z_1) \neq h(z_2)$. It has a successful *execution* $I'$ with $h'$, in symbols $I \xrightarrow{\sigma,h} I'$, if

(1) any $\bar{z} \in$ *eq-class*$(\sigma)$ and $z_1, z_2 \in \bar{z}$ such that $h(z_1) \neq h(z_2)$ satisfy $h(z_1), h(z_2)$ are both literals, or $h(z_1), h(z_2)$ are both non literals, and

(2) $I' = h'(I)$ and $h'$ is a homomorphism such that $dom(h') = h(dom(h)) \cap \mathsf{Null}$ i.e., $h'$ assigns values to the null values used by $h$, and for any $\bar{z} \in$ *eq-class*$(\sigma)$ and any $z_1, z_2 \in \bar{z}$ we have $h'(h(z_1)) = h'(h(z_2))$.

If $\sigma$ is triggered in $I$ by $h$ but does not have a successful execution, we say that it *fails*, in symbols $I \xrightarrow{\sigma,h} \bot$.

### A.1.1 Value consistency

We prove here the different propositions made in Section 5.1.

**Proof of Lemma 6**
We show the left-to-right direction. We claim that (1) for any instance $I$ of $\mathbf{R}$, a solution for $I$ to $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ is included in a solution for $I$ to $\mathscr{E}$.

We now show (1). Take a universal solution $J$ to $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ and a universal solution $J'$ to $\mathscr{E}$. We prove that $J \subseteq J'$. We fix two chase sequence $s$ and $s'$ such that the instance where there can not be triggered more rules are $J$ and $J'$ for $s$ and $s'$ respectively. Since $s'$ is finite, i.e. there is not failure, then the egds only are triggered for those triples that contain null as objects. This is not produced by $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$. Since $s'$ has applied the rules in $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$, then $J$ is in $J'$.

Now, we take a pair $(T_n, f_n) \in \mathscr{T} \times \mathscr{F}$. Assume $(T_n, f_n)$ is accessible in $\mathscr{E}$. Then, we construct an instance $I$ of $\mathscr{R}$ where $\mathscr{R} = \{R_1, \ldots, R_n\}$ such that $dom(I) = \{b\}$ and for each $R \in \mathscr{R}$ is of arity $n \in \mathbb{N}$. Because $(T_n, f_n)$ is accessible, we know that there is a sequence $\sigma_0, \ldots, \sigma_i$ of st-tgds s.t.:

- $head(\sigma_0) = T_0(f_0(\bar{y_0}))$, and
- $head(\sigma_i) = Triple(f_{i-1}(\bar{x_i}), p_i, f_i(\bar{y_i}))$ for any $1 \leq i \leq n$, and
- $(p :: T_i)^{\mu} \in \delta(T_{i-1})$ for some multiplicity $\mu$, and
- $(T, f, p) = (T_n, f_n, p_n)$

for some type symbols $\{T_i \mid 0 \leq i < n\} \subseteq \mathscr{T}$, function symbols $\{f_i \mid 0 \leq i < n\} \subseteq \mathscr{F}$, and IRIs $\{p_i \mid 1 \leq i < n\} \subseteq \mathsf{Iri}$.

Now, we take any solution $J$ for $I$ to $\mathscr{E}$. By chasing $I$ with the sequence of st-tgds $\sigma_0, \ldots, \sigma_i$ and $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$ of triple constraints from accessibility of $(T_n, f_n)$, we

have that

$$A = \{T(f_0(\bar{b})), \mathit{Triple}(f_0(\bar{b}), p_1, f_1(\bar{b})), T_1(f_1(\bar{b})), \mathit{Triple}(f_1(\bar{b}), p_2, f_2(\bar{b})), \ldots, \\ \mathit{Triple}(f_{n-1}(\bar{b}), p_n, f_n(\bar{b})), T_n(f_n(\bar{b}))\}$$

Since chase sequence of $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ is finite, then there is an instance $A$ where there is no rule triggered. By chase sequence definition, $A$ is a universal solution to $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$. By definition of universal solution, there is an homomorphism $h : A \to J'$ such that $h(c) = c$ for all $c \in \mathit{dom}(A)$ and $J'$ a solution to $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$. Since in $A$ there are no nulls, $A \subseteq J'$. By claim (1) and $A \subseteq J'$, $A \subseteq J$. Let $a = f_n^F(\bar{b})$, i.e. $a \in \mathit{ran}(f_n^F)$. Then $T_n(a) \in J$. Thus, $T_n(a)$ is in all solutions for $I$ to $\mathscr{E}$.

Now, we show the right-to-left direction. We take a pair $(T_n, f_n)$ and assume there exist an instance $I$ of $\mathscr{R}$ and a constant $a$ in $\mathit{ran}(f_n^F)$ s.t. $T_n(a)$ is a fact in all solutions for $I$ to $\mathscr{E}$.

We have to prove that $(T_n, f_n)$ is accessible in $\mathscr{E}$. We fix a chase sequence $s = I = J_0 \xrightarrow{\sigma_1, h_1} J_1 \xrightarrow{\sigma_2, h_2} \ldots J_{m-1} \xrightarrow{\sigma_m, h_m} J_m$ where $\sigma_i$ a dependency in $\Sigma_{\text{st}}$ and $h_i : \sigma_i \to J_i$ an homomorphism for any $i \in \{1, \ldots, m\}$.

We claim (1) that for any finite chase sequence $s$ for dependencies $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$, any $a \in \mathsf{Const}$, any $T \in \mathscr{T}$, any $k \in \{1, \ldots, |s|\}$, if $J_k$ contains the fact $T(a)$ then there is $f \in \mathscr{F}$ such that $(T, f)$ is accessible in $\mathscr{E}$.

We claim (2) that for any finite chase sequence $s$ for dependencies $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$, any $a \in \mathsf{Const}$, any $T \in \mathscr{T}$, any $k \in \{2, \ldots, |s|\}$, if $J_k$ contains the fact $T(a)$ then there is $k' < k$ and $T' \in \mathscr{T}$ and $p \in \mathsf{Pred}$ and $a' \in \mathsf{Const}$ such that $J_{k'}$ contains the facts $T'(a')$, $\mathit{Triple}(a', p, a)$ and $p :: T''^{\mu} \in \delta(T')$ for some $\mu$.

We claim (3) for any finite chase sequence $s$, any $k \in \{1, \ldots, |s|\}$, any $b, b' \in \mathsf{Const}$ and any $q \in \mathsf{Iri}$ if $J_k$ contains the fact $\mathit{Triple}(b, q, b')$ then there is a st-tgd $\sigma \in \Sigma_{\text{st}}$ such that $\mathit{head}(\sigma) = \mathit{Triple}(f(\bar{y_1}), q, g(\bar{y_2}))$ for some $f, g \in \mathscr{F}$ and $\bar{y_1}$ and $\bar{y_2}$ in $\mathit{vars}(\sigma)$; and there is an homomorphism $h : \mathit{head}(\sigma) \to J_k$ such that $h(\bar{y_1}) = b$ and $h^F(\bar{y_2}) = b'$.

We now prove claim (1). Let $\sigma$ be a rule in $\Sigma_{\text{st}}$ such that $\mathit{head}(\sigma) = T(f(\bar{x}))$ for some $\bar{x} \in \mathit{vars}(\sigma)$. Let $h : \sigma \to J_k$ such that $h(f(\bar{x})) = a$. Since $J_k$ contains a fact produced in chase step of $s$, then this $h$ was triggered in some instance before $J_k$. By definition of accessibility, we conclude that $(T, f)$ is accessible in $\mathscr{E}$.

Next, we prove claim (2). Let $\sigma$ be a rule in $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$ such that $\mathit{head}(\sigma) = T(x)$. Let $h : \sigma \to J_k$ such that $T(h(x)) = a$. Since $J_k$ contains a fact produced in chase step of $s$, then $\sigma$ exists with $h$ because it was triggered in some instance before $J_k$. Let that instance be $J_{k'}$ such that $k' < k$. Thus, $T'(h(x))$, $\mathit{Triple}(h(x), p, h'(y))$ in $J_{k'}$ for some $p \in \mathsf{Iri}$. Let $h(x) = a'$ for some $a' \in \mathsf{Const}$. Since $\sigma \in \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ then $p :: T \in \delta(T')$.

Finally, we prove claim (3). Assume $J_k$ contains the fact $\mathit{Triple}(b, q, b')$. Since $J_k$ is an instance of chase sequence at $k$ chase step of $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}$, then the only rule that can be triggered in some instance before $J_k$ is of the form $\varphi \Rightarrow \mathit{Triple}(f(\bar{y_1}), q, g(\bar{y_2}))$ for some $\bar{y_1}$ and $\bar{y_2}$ in $\mathit{vars}(\varphi)$. Thus, we conclude that there is an homomorphism $h$ such that $h(f(\bar{y_1})) = b$ and $h^F(q) = q$ and $h(f(\bar{y_2})) = b'$.

Since the fact $T_n(a) \in J_m$ is the result of either trigger a rule in $\Sigma_{\text{st}}$ or $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$, we can apply claim (1) or (2) respectively. Considering the rule be in $\Sigma_{\text{st}}$ and by claim (1), we have that $(T_n, f_n)$ is accessible. Considering the rule be in

$\Sigma_{\mathbf{S}}^{\mathbf{TP}}$, we obtain that there is a $k' < m$ and $T' \in \mathscr{T}$ and $a' \in \mathsf{Const}$ and $p \in \mathsf{Pred}$ such that the facts $T'(a')$ and $\mathit{Triple}(a', p, a)$ are in $J_{k'}$ and $p :: T_n^{\mu} \in \delta(T')$ for some $\mu$. Applying claim (3) to the fact $\mathit{Triple}(a', p, a) \in J_{k'}$, we have that $\sigma = \mathit{Triple}(f(\bar{y_1}), p, f_n(\bar{y_2}))$ for some $f \in \mathscr{F}$ such that $a' \in \mathit{ran}(f^F)$. Let this $f$ be $f_{n-1}$. We analyze the fact $T'(a')$ as in the beginning with claims (1) or (2). Assume that claim (1) is not applied until we are in step 2. Then we have only applied claim (2) and (3) sequentially obtaining at this chase step that $T_0(a_0)$, $\mathit{Triple}(a_0, p_1, a_1)$ in $J_2$. By applying claim (1) to the fact $T_0(a_0)$ we obtain a rule $\sigma_0 = T_0(f_0(\bar{y_0}))$. This rule together with the set of rules $\sigma_i, \ldots, \sigma_j$ where $i < j < m$ that are in $\Sigma_{\mathrm{st}}$, which were obtained by the application of claim (3) allows to conclude that $(T_n, f_n)$ is accessible in $\mathscr{E}$.

**Proof of Proposition 7**

Suppose first that $\pi, \sigma, \sigma', h$ are as in the premise of 1. and let $\pi = \sigma_0, \ldots, \sigma_n$, $\mathit{head}(\sigma) = \mathit{Triple}(f(\bar{z}), p, t)$, $\mathit{head}(\sigma') = \mathit{Triple}(f(\bar{z'}), p, t')$ and the $\sigma_0, \ldots, \sigma_n$ as in Definition 5, so $(T, f, p) = (T_n, f_n, p_n)$. Because $(T_n, f_n)$ is accessible in $\mathscr{E}$ with $\pi$, we know that $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$ contains the rules $\mathbf{TP}(T_{i-1}, p_i, T_i)$ for any $0 < i \leq n$. Let $\mathbf{TP}(T_{i-1}, p_i, T_i) = T_{i-1}(u_i) \wedge \mathit{Triple}(u_i, p_i, v_i) \Rightarrow T_i(v_i)$ for any $0 < i \leq n$, where w.l.g. $u_i, v_i$ are fresh w.r.t. the variables used in $\sigma_0, \ldots, \sigma_n, \sigma, \sigma'$ and $\{u_i, v_i\}$ is disjoint from $\{u_j, v_j\}$ whenever $i \neq j$. Let $h' = h \circ h_{\pi,\sigma,\sigma'}$, then by definition of $I_{\pi,\sigma,\sigma'}$ and $h$ it follows that $I$ is the disjoint union of $h'(B_{\pi,\sigma,\sigma'})$ and $I'$, the latter containing the facts of $I'$ that are not images by $h$ of some fact in $I_{\pi,\sigma,\sigma'}$.

Consider the following chase sequence $s$ starting at $I$; note that in the sequel we abuse the notation and use $h'$ as its restriction on any subset of variables in its domain. The first chase step of $s$ is $I \xrightarrow{\sigma_0, h'} I_0$. The subsequent chase steps are defined inductively by adding the following two chase steps for all $0 < i \leq n$:

$$I_{i-1} \xrightarrow{\sigma_i, h'} I'_i \xrightarrow{\mathbf{TP}(T_{i-1}, p_i, T_i), h_i} I_i,$$

where $h_i$ is defined by $h_i(u_i) = h'(f_{i-1}(\bar{y}_{i-1}))$ and $h_i(v_i) = h'(f_i(\bar{y}_i))$.

Thus $s$ is of the form:

$$I \xrightarrow{\sigma_0, h'} I_0 \xrightarrow{\sigma_1, h'} I'_1 \xrightarrow{\mathbf{TP}(T_0, p_1, T_1), h_1} I_1 \to \cdots \to I_{n-1} \xrightarrow{\sigma_n, h'} I'_n \xrightarrow{\mathbf{TP}(T_{n-1}, p_n, T_n), h_n} I_n.$$

We now show that $s$ is indeed a chase sequence. That is, we need to show that the homomorphism of each step above is indeed a homomorphism from the body of the dependency being applied to the instance to which the step is applied. It immediately follows from the definitions and hypotheses that (1) $I_0 = I' \cup h'(B_{\pi,\sigma,\sigma'}) \cup T_0(h(f_0(\bar{y}_0)))$ where $I'$ contains the facts of $I$ that are not images of some fact of $I_{\pi,\sigma,\sigma'}$ by $h$. For any $1 \leq i \leq n$ we show the following by induction on $i$:

(2) $I'_i = I_{i-1} \cup h'(\mathit{head}(\sigma_0) \cup \cdots \cup \mathit{head}(\sigma_i))$;

(3) $I_i = I'_i \cup T_i(h'(f_i(\bar{y}_i)))$.

For the base case $i = 1$. From (1) it follows that $h' : \sigma_1 \to I_0$ is a homomorphism, and by definition of the chase, applying this homomorphism on $I_0$ yields $I'_1 = I_0 \cup h'(\mathit{head}(\sigma_1))$, thus (2) holds. Now from (1) and (2) we know that $I'_1$ contains the facts $T_0(h'(f_0(\bar{y}_0)))$ and $\mathit{Triple}(h'(f_0(\bar{x}_1)), p_1, f_1(\bar{y}_1)) = h'(\mathit{head}(\sigma_1))$. Recall that by definition, $h_{\pi,\sigma,\sigma'}(\bar{x}_1) = h_{\pi,\sigma,\sigma'}(\bar{y}_0)$, so also $h'(\bar{x}_1) = h'(\bar{y}_0)$, thus $h_1$

is indeed an homomorphism from $T_{i-1}(u_i) \wedge \mathit{Triple}(u_i, p_i, v_i)$ into $I'_i$ and the resulting instance is indeed $I'_i \cup T_i(h'(f_i(\bar{y}_i)))$.

The same arguments apply for the induction step for showing that $h' : \sigma_i \to I_{i-1}$ and $h_i : \mathbf{TP}(T_{i-1}, p_i, T_i)$ are homomorphisms, and their application yields the instances described in (2) and (3).

Consider now the chase sequence

$$s' = I_n \xrightarrow{\sigma, h} I_\sigma \xrightarrow{\sigma', h} I_{\sigma'}.$$

It immediately follows from the definition of $h$, from (3) and from the definition of a chase step that $I_\sigma = I_n \cup \{\mathit{Triple}(h(f_n(\bar{y}_n)), p_n, h(t))\}$ and $I_{\sigma'} = I_\sigma \cup \{\mathit{Triple}(h(f_n(\bar{y}_n)), p_n, h(t'))\}$.

Finally, consider any terminating chase sequence by $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ starting at $I_{\sigma'}$, and let $J'$ be its terminal instance; we know that such finite chase instance exists because the dependencies in $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ are full. Then $J'$ is a universal solution for $I$ to $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ and moreover $I_{\sigma'} \subseteq J'$, so $\{T_n(h'(f_n(\bar{y}_n))), \mathit{Triple}(h'(f_n(\bar{y}_n)), p_n, h'(t)), \mathit{Triple}(h'(f_n(\bar{y}_n)), p, h'(t'))$ $J' \subseteq J$ for any $J$ solution for $I$ to $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$. We conclude the proof of the left-to-right direction of 1 remarking that by definition, $h_{\pi,\sigma,\sigma'}(\bar{y}_n) = h_{\pi,\sigma,\sigma'}(\bar{z}) = h_{\pi,\sigma,\sigma'}(\bar{z}')$, so also $h'(\bar{y}_n) = h'(\bar{z}) = h'(\bar{z}')$. This also shows 2. in the case where the left-to-right direction of 1. holds.

**Proof of Theorem 8** We first show the left-to-right direction by proving its contraposition. Let a violation sort $(T, f, p)$, and let $\pi, \sigma, \sigma', J, h$ be as in the theorem, in particular $h \circ h_{\pi,\sigma,\sigma'}(t) \neq h \circ h_{\pi,\sigma,\sigma'}(t')$. Then by Proposition 7 we have that $J_0$, the core pre-solution for $I$ to $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$, includes $w = \{T(a), \mathit{Triple}(a, p, b), \mathit{Triple}(a, p, b')\}$, where $a = h \circ h_{\pi,\sigma,\sigma'}(f(\bar{z})) = h \circ h_{\pi,\sigma,\sigma'}(f(\bar{z}'))$, $b = h \circ h_{\pi,\sigma,\sigma'}(t)$ and $b' = h \circ h_{\pi,\sigma,\sigma'}(t')$. By hypothesis, $b \neq b'$, so $w$ is a $(T, f, p)$-violation, so the core pre-solution of $I$ does not satisfy $\Sigma_{\mathbf{S}}^{\mathbf{PF}}$.

We show the right-to-left direction again proving its contraposition.

Suppose there exists a consistent source instance $I$ s.t. its core pre-solution $J_0$ includes a $(T, f, p)$-violation, say $w = \{T(a), \mathit{Triple}(a, p, b), \mathit{Triple}(a, p, b')\}$ with $b \neq b'$ and $(T, f, p)$ a violation sort. So Proposition 7 applies allowing to deduce that there exist $\pi, \sigma, \sigma', h$ s.t. $(T, f)$ is accessible with $\pi$ in $\mathscr{E}$, $\sigma, \sigma'$ are $(T, f, p)$-contentious st-tgds, and $h : I_{\pi,\sigma,\sigma'} \to I$ is a homomorphism, with $\mathit{head}(\sigma) = \mathit{Triple}(f(\bar{z}), p, t)$ and $\mathit{head}(\sigma') = \mathit{Triple}(f(\bar{z}'), p, t')$, and $a = h \circ h_{\pi,\sigma,\sigma'}(f(\bar{z})) = h \circ h_{\pi,\sigma,\sigma'}(f(\bar{z}'))$, $b = h \circ h_{\pi,\sigma,\sigma'}(t)$ and $b' = h \circ h_{\pi,\sigma,\sigma'}(t')$. Therefore $h \circ h_{\pi,\sigma,\sigma'}(t) \neq h \circ h_{\pi,\sigma,\sigma'}(t')$. Suppose by contradiction that for all $J, h'$ s.t. $J$ solution for $I_{\pi,\sigma,\sigma'}$ to $\Sigma_{\text{fd}}$ and $h' : I_{\pi,\sigma,\sigma'} \to J$ the corresponding homomorphism we have $(h_{\pi,\sigma,\sigma'} \circ h')(t) = (h_{\pi,\sigma,\sigma'} \circ h')(t')$. We are then able to construct a contradiction to the fact that $I$ is a consistent source instance that satisfies the source functional dependencies.

### A.1.2 Node kind consistency

We start with a slight generalization of the construction of a universal simulation solution presented in Section 6 that will be used for computing $\mathit{CoTypes}(J)$ and for the proofs of Lemma 10 and Lemma 11. We recall the definitions from Section 6 are useful for this definition.

Fix a source instance $I$, and suppose that $J$ is a solution for $I$ to $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}} \cup \Sigma_{\mathbf{S}}^{\mathbf{PF}}$. The types associated to a node in a typed RDF graph $\mathit{types}_G(n) = \{T \mid T(n) \in$

$G\}$. The IRIs required by a set of types $X$ is the set $Req(X) = \{p \mid \exists S, \mu, T \in X.\ p :: S^{\mu} \in \delta(T)\}$. The frontier of $J$ is the following set

$$\mathbb{F} = \{(n, p) \mid n \in nodes(J),\ p \in Req(types_J(n)),\ \nexists m.\ Triple(n, p, m) \in J\}.$$

For any set of types $X$ and IRI $p$, $\Delta(X, p)$ is the set of types that must hold at any node having a type from $X$ and reachable by a $p$-labeled edge:

$$\Delta(X, p) = \{S \mid p :: S^{\mu} \in \delta(T) \text{ for some } T \in X \text{ and } \mu \in \{\mathtt{?}, \mathtt{1}, \mathtt{+}, \mathtt{*}\}\}.$$

Whether $J$ can be augmented to a solution that satisfies $\Sigma_{\mathbf{S}}$ depends on the sets of types that co-occur in the frontier of $J$. Define the set of subsets of $\mathscr{T} \cup \{Literal\}$: $N_0 = \{\Delta(types_J(n), p) \mid (n, p) \in \mathbb{F}\}$. Then let $N_J = \bigcup_{i=0}^{\infty} N_i$, where $N_i = \{\Delta(X, p) \mid X \in N_{i-1},\ p \in Req(X)\}$ for any $i \geq 1$. This process reaches a fixed point in a final number of steps.

**Lemma 26** *For any typed graph $J$, $N_J = CoTypes(J)$.*

As a corollary of Lemma 26 we get that $CoTypes(J)$ can be effectively computed.

### Proofs of Lemma 10 and Lemma 11

**Lemma 27** *For any instance $I$ of $\mathscr{R}$ and any $J$ solution for $I$ to $\Sigma_{\mathrm{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}} \cup \Sigma_{\mathbf{S}}^{\mathbf{PF}}$, if $N_J$ contains a set $X$ with $\{Literal, T\} \subseteq X$ for some type $T$ in $\mathscr{T}$, then $I$ does not admit a solution to $\mathscr{E}$ that includes $J$.*

*Proof.* [Sketch of proof] We first show that if $X$ is a set in $N_J$, then any solution $G$ for $I$ to $\mathscr{E}$ that includes $J$ must contain a node $n_X$ with $X \subseteq types_G(n_X)$. This is done by induction on the index $i$ s.t. $X \in N_i$. Next we show that if *Literal* is in $X$, then $n_X$ must be a literal, and if some type $T$ from $\mathscr{T}$ is in $X$, then $n_X$ must be an IRI or a blank node. Then the lemma follows by contradiction.

Remark that Lemma 10 is an immediate consequence of Lemma 26 and Lemma 27.

Now, if $N_J$ does not contain any set $X$ in which *Literal* co-occurs with some type $T$ from $\mathscr{T}$, then we can construct a solution for $I$ to $\mathscr{E}$ that includes $J$, as follows. For any $X \in N_J$ s.t. $X \subseteq \mathscr{T}$, let $n_X$ be a fresh blank node, i.e. $n_X \in \mathsf{NullIri} \setminus dom(J)$. For any $X \in N_J$ and $p \in Req(X)$, let $n_{X,p}$ be a fresh null literal, i.e. $n_{X,p} \in \mathsf{NullLit} \setminus dom(J)$. Define the graph $G_{\mathbf{S}}$ as follows.

$$\begin{aligned} G_{\mathbf{S}} = &\{Triple(n, p, n_X) \mid (n, p) \in \mathbb{F} \wedge X = \Delta(types_J(n), p) \subseteq \mathscr{T}\} \cup \\ &\{Triple(n_X, p, n_{X,p}) \mid (n, p) \in \mathbb{F} \wedge \Delta(types_J(n), p) = \{Literal\}\} \cup \\ &\{Triple(n_X, p, n_{X'}) \mid X \in N_J \wedge p \in Req(X) \wedge X' = \Delta(X, p) \subseteq \mathscr{T}\} \cup \\ &\{Triple(n_X, p, n_{X,p}) \mid X \in N_J \wedge p \in Req(X) \wedge \Delta(X, p) = \{Literal\}\} \cup \\ &\{T(n_X) \mid X \in N_J \wedge T \in X\}. \end{aligned}$$

**Lemma 28** *For any source instance $I$ and any $J$ solution for $I$ to $\Sigma_{\mathrm{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}} \cup \Sigma_{\mathbf{S}}^{\mathbf{PF}}$, if $N_J$ does not contain a set $X$ with $\{Literal, T\} \subseteq X$ for some type $T$ in $\mathscr{T}$, then $J \cup G_{\mathbf{S}}$ is a solution for $I$ to $\mathscr{E}$.*

*Proof.* [Sketch of proof.] $J \cup G_{\mathbf{S}}$ satisfies $\Sigma_{\text{st}}$ as $J$ does. It is easy to see by its definition that $J \cup G_{\mathbf{S}}$ also satisfies the **TP** and **PE** dependencies in $\Sigma_{\mathbf{S}}$. Regarding the **PF** dependencies in $\Sigma_{\mathbf{S}}$: on the one hand, $J$ satisfies the **PF** dependencies by hypothesis. On the other hand, by construction, the triples added in $G_{\mathbf{S}}$ are such that no node has more than one $p$-outgoing edge for any IRI $p$. Therefore $J \cup G_{\mathbf{S}}$ does not contain a trigger for a **PF** dependency. We point out that Lemma 11 is an immediate consequence of Lemma 26 and Lemma 28.

**Proof of Theorem 12**

Let $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$ and let $\mathbf{S} = (\mathscr{T}, \delta)$. We first show that it is decidable whether $\mathscr{E}$ is node kind consistent.

Let $N_0$ be the set of subsets of $\mathscr{T} \cup \{\mathit{Literal}\}$ such that $X \in N_0$ iff there exists a function symbol $f \in \mathscr{F}$ s.t. $X = \{T \mid (T, f) \text{ accessible in } \mathscr{E}\}$. Then we define $\mathit{CoTypes}(\mathscr{E}) = \bigcup_{i=0}^{\infty} N_i$, where $N_i = \{\Delta(X, p) \mid X \in N_{i-1},\ p \in \mathit{Req}(X)\}$ for any $i \geq 1$. Note that $\mathit{CoTypes}(\mathscr{E})$ converges to a fix point in a finite number of steps.

**Lemma 29** *$\mathscr{E}$ is node kind consistency iff $\mathit{CoTypes}(\mathscr{E})$ does not contain a set $X$ s.t. $\{T, \mathit{Literal}\} \subseteq X$ for some $T \in \mathscr{T}$.*

*Proof.* We need to show that for any $I$ instance of $\mathscr{R}$ $\mathit{CoTypes}(J_0)$ does not contain a set $X$ with $\{T, \mathit{Literal}\} \subseteq X$ for some $T \in \mathscr{T}$ iff $\mathit{CoTypes}(\mathscr{E})$ does not contains a set $X$ s.t. $\{T, \mathit{Literal}\} \subseteq X$ for some $T \in \mathscr{T}$.

For the left-to-right direction we show that if $\mathit{CoTypes}(\mathscr{E})$ contains such $X$, then there exists an instance $I$ s.t. $\mathit{CoTypes}(J)$ contains a set $X'$ with $X \subseteq X'$. It is enough to take the $I$ such that it contains exactly one fact for any relation in $\mathscr{R}$, and such that $\mathit{dom}(I) = \{b\}$ for some constant $b$.

For the right-to-left direction, define the graph $R$ which vertices are $\mathscr{T}$ and that has an edge labelled with $p$ from $T_1$ to $T_2$ iff $\delta(T_1)$ contains a triple constraint $p :: T_2^1$ or $p :: T_2^+$. Intuitively, a $p$-edge from $T_1$ to $T_2$ in $R$ indicates that the label $p$ is required in every node that has type $T_1$, and every $p$-edge leads to a node that must have type $T_2$.

We show that if there is an instance $I$ and a type $T \in \mathscr{T}$ s.t. $\mathit{CoTypes}(J_0)$ contains $X$ with $\{T, \mathit{Literal}\} \subseteq X$ then necessarily the frontier of $J_0$ contains some $(n, p)$ s.t. $n$ is not null and there exist types $S$, resp. $S'$ in $\mathit{types}_{J_0}(n)$ that are, intuitively, the reasons why $T$, resp. $\mathit{Literal}$, were added to $X$ during the construction of $\mathit{CoTypes}(J_0)$. Note that such $S, S'$ are not necessarily distinct. Moreover, there is a sequence $w$ of IRI's and a path in $R$ from $S$ to $T$ labelled with $w$ and a path in $R$ from $S'$ to $\mathit{Literal}$ labelled with $w$.

Using Lemma 6 we deduce that $(S, f)$ and $(S', f)$ are accessible in $\mathscr{E}$, where $f$ is the function symbol s.t. $n \in \mathit{ran}(f^F)$. Then we show inductively on the $w$ that during the construction of $\mathit{CoTypes}(\mathscr{E})$ we will reach a set $X'$ that contains both $T$ and $\mathit{Literal}$.

We now describe a coNP decision procedure for $\mathscr{E}$ being node kind consistent. A certificate for a node kind inconsistency is composed of types $T, S, S'$ and a function symbol $f \in \mathscr{F}$ as in the proof of the right-to-left direction of Lemma 29. More precisely, choose non-deterministically $T, S, S'$ and $f$ s.t. $(S, f)$ and $(S', f)$ are accessible in $\mathscr{E}$ (the latter can be tested in polynomial time). According to the proof of Lemma 29, it is enough to test whether there exists a sequence $w$ of IRIs s.t. $R$ has paths labelled with $w$ from $S$ to $T$ and from $S'$ to $\mathit{Literal}$.

The latter can be polynomially tested by considering two finite state automata $A_S$ and $A_{S'}$ that are both derived from the graph $R$. That is, both automata have the vertices of $R$ as states and the edges of $R$ as transitions. $A_S$ has $S$ as initial state, while $S'$ is the initial state of $A_{S'}$. We then compute the product automaton $A_S \times A_{S'}$ in polynomial time. Then there exists $w$ as above iff state $(T, \mathit{Literal})$ is accessible in the product automaton. In this case, $w$ is the shortest path from $(S, S')$ to $(T, \mathit{Literal})$ in this automaton.

Now the proof of Theorem 12 can be completed:

- If $I$ is an instance of $\mathbf{R}$ and the core pre-solution for $I$ to $\mathscr{E}$ is value consistent and node kind consistent, then $I$ admits a solution to $\mathscr{E}$ (Lemma 11).
- If $I$ is an instance of $\mathbf{R}$ and the core pre-solution of $I$ to $\mathscr{E}$ is not value consistent, resp. is not node kind consistent, then $I$ does not admit a solution to $\mathscr{E}$ (corollary of Theorem 8, resp. Lemma 10).
- It is decidable whether $\mathscr{E}$ is value consistent (Lemma 30) and it is decidable whether $\mathscr{E}$ is node kind consistent (here above).

### A.1.3 Proof of Theorem 13

**Upper bound**

Checking node kind consistency is in co-NP as shown in Section A.1.2.

Regarding value consistency:

**Lemma 30** *Deciding whether $\mathscr{E}$ is value consistent is in coNP.*

*Proof.* The contraposition of Theorem 8 implies that $\mathscr{E}$ is value inconsistent iff there exists a source instance ($J$ in the theorem) that satisfies the source integrity constraints $\Sigma_{\text{fd}}$ but is value inconsistent. This gives a co-NP decision procedure for value consistency. Indeed, $J$ as in the theorem is a certificate for the value inconsistency. We now argue that such certificate has size polynomial in the size of $\mathscr{E}$ and we can test in polynomial time whether it is indeed value inconsistent. First, guess a violation sort $(T, f, p)$, an elementary sequence $\pi = \sigma_0, \ldots, \sigma_n$ and two st-tgds $\sigma, \sigma'$ from $\Sigma_{\text{st}}$. This is done in polynomial time as $\pi$ is elementary. Then check that $(T, f)$ is accessible in $\mathscr{E}$ with $\pi$ and that $\sigma, \sigma'$ are contentious with sort $(T, f, p)$ and construct the source instance $I_{\pi,\sigma,\sigma'}$. This step is done in polynomial time as well. Finally, chase $I_{\pi,\sigma,\sigma'}$ with $\Sigma_{\text{fd}}$. The latter can also be done in polynomial time because the instance $I_{\pi,\sigma,\sigma'}$ has a polynomial size, and all bodies of dependencies in $\Sigma_{\text{fd}}$ contain exactly two atoms, thus require to compute a unique join in order to be evaluated. Additionally, $\Sigma_{\text{fd}}$ chase steps do not increase the size of the instance, and only a polynomial number of chase steps can be executed before a solution or a failure is reached. The result of the chase is the certificate $J$. Consequently, deciding whether $\mathscr{E}$ is value consistent is in coNP.

**Lower bound** We prove coNP-hardness with reduction from the complement of SAT. Take any CNF

$\varphi = c_1 \wedge \ldots \wedge c_m$, where $c_j = \ell_{j,1} \vee \ldots \vee \ell_{j,k_j}$ is a clause over the variables $x_1, \ldots, x_n$. We construct the corresponding data exchange setting $\mathscr{E}_\varphi$ as follows.

The relational schema consists of the following binary relation names (each having the first attribute as a key $A \to B$)

$$V_{\mathbf{t}}(\underline{A}, B), V_{\mathbf{f}}(\underline{A}, B), R_1(\underline{A}, B), \ldots, R_m(\underline{A}, B)$$

The constructor set is

$$\mathscr{F} = \{f_1, \ldots, f_m, f_{m+1}\}$$

and their implementation is very straightforward $f_i(x) = "i:" + \mathtt{str}(x)$. We use the types

$$\mathscr{T} = \{T_1, \ldots, T_m, T_{m+1}\}$$

and the shape constraints:

$$T_j \to a :: T^*_{j+1} \qquad \text{for } 1 \leq j \leq m \text{ and} \tag{1}$$
$$T_{m+1} \to a :: \mathit{Literal}^{\mathbf{1}}. \tag{2}$$

The source to target dependencies are as follows. First, we have the two rules:

$$V_{\mathbf{t}}(x, y) \Rightarrow \mathit{Triple}(f_{m+1}(x), a, y) \tag{3}$$
$$V_{\mathbf{f}}(x, y) \Rightarrow \mathit{Triple}(f_{m+1}(x), a, y) \tag{4}$$

Next, for any $1 \leq j \leq m$ let $c_j = \ell_{j,1} \vee \ldots \vee \ell_{j,k_j}$ and for $q \leq k \leq k_j$ if $\ell_{j,k} = x_i$, then we add this rule

$$R_i(x, y) \wedge V_{\mathbf{t}}(x, y) \Rightarrow \mathit{Triple}(f_j(x), a, f_{j+1}(x)) \tag{5}$$

and otherwise if $\ell_{j,k} = \neg x_i$, then we add this rule

$$R_i(x, y) \wedge V_{\mathbf{f}}(x, y) \Rightarrow \mathit{Triple}(f_j(x), a, f_{j+1}(x)) \tag{6}$$

And finally, we add the following two rules:

$$V_{\mathbf{t}}(x, y) \Rightarrow T_1(f_1(x)) \tag{7}$$
$$V_{\mathbf{f}}(x, y) \Rightarrow T_1(f_1(x)) \tag{8}$$

We claim that

$$\varphi \in \text{SAT} \quad \text{iff} \quad \mathscr{E}_\varphi \text{ is not consistent.}$$

For *only if* part, we take a valuation $V$ that satisfies $\varphi$ and construct an instance $I_V$ as follows. We fix 3 constants $c$, $\mathbf{t}$, and $\mathbf{f}$. The instance is

$$I_V = \{V_{\mathbf{t}}(c, \mathbf{t}), V_{\mathbf{f}}(c, \mathbf{f})\} \cup \{R_i(c, \mathbf{t}) \mid i \in \{1, \ldots, n\},\ V(x_i) = \mathbf{true}\} \cup \{R_i(c, \mathbf{f}) \mid i \in \{1, \ldots, n\},\ V(x_i) = \mathbf{false}\}.$$

It is easy to see that $I_V$ is consistent and with a simple inductive proof we can show that the result of chase on $I_V$ contains $T_{m+1}(f_{m+1}(c))$ and the two triples $\mathit{Triple}(f_{m+1}(c), a, \mathbf{t})$ and $\mathit{Triple}(f_{m+1}(c), a, \mathbf{f})$ which violates the shape constraint on the type $T_{m+1}$.

For the *if* part, we take a consistent instance $I$ such that chase of $I$ with $\mathscr{E}_\varphi$ is equal to $J$ and violates the shape constraints. The only shape constraint that

can be violated is the constraint on the type $T_{m+1}$ (all remaining constraints can be satisfied by chase by adding null values if needed). Consequently $J$ contains $T_{m+1}(t_{m+1})$, $\mathit{Triple}(t_{m+1}, a, \mathbf{t})$, and $\mathit{Triple}(t_{m+1}, a, \mathbf{f})$, for some $t_{m+1}$, $\mathbf{f}$, and $\mathbf{t}$. Naturally, the two triples must be introduced with the rules (3) and (3), and therefore, there is a constant $c$ such that $t_{m+1} = f_{m+1}(c)$, $V_{\mathbf{t}}(c, \mathbf{t}) \in I$, and $V_{\mathbf{f}}(c, \mathbf{f}) \in I$. Furthermore, with a simple inductive proof we can show that for every $j \in \{1, \ldots, m\}$ we have $T_j(f_j(c)) \in J$, $\mathit{Triple}(f_j(c), a, f_{j+1}(c)) \in J$. We observe the triples $\mathit{Triple}(f_j(c), a, f_{j+1}(c))$ can be only added by chase with the use of rules (5) and (6), and the inductive proof also shows that every clause $c_j$ has at least one literal for which the corresponding rule must have been triggered. Since $I$ is consistent for no $i \in \{1, \ldots, n\}$ can $I$ have both $R_i(c, \mathbf{t})$ and $R_i(c, \mathbf{f})$ ($I$ may have none of the two). We can therefore define the following valuation

$$V(x_i) = \begin{cases} \mathbf{true} & \text{if } R_i(c, \mathbf{t}) \in I, \\ \mathbf{false} & \text{otherwise.} \end{cases}$$

We show that $V$ satisfies $\varphi$ by observing that if for the chase triggers a clause (5) or (6) that corresponds to some literal $\ell$ of $c_j$, then $V$ satisfies $c_j$. We finish the proof by observing that the proposed reduction is polynomial.

## A.2 Consistency of Non-Constructive st-tgds

Using the notations from Section 5.3, recall that any pair of rules in $\Sigma_{\text{st}}$ use pairwise disjoint variables, $\theta$ is the set of terms that appear in the heads of $\Sigma_{\text{st}}$, and $\mathscr{T}$ is the set of type names of the shapes schema $\mathbf{S}$. Denote $V$ the universally quantified variables that appear in $\Sigma_{\text{st}}$. We write $A \in \Sigma_{\text{st}}$ when the atom $A$ appears in some head in $\Sigma_{\text{st}}$, thus $\Sigma_{\text{st}}$ is viewed as a monadic relation over atoms. We use $t, t', u, u'$ to denote terms, and $x, x', y, y'$ and $\bar{x}, \bar{y}$ to denote variables and vectors of variables, respectively. The relations $\mathit{Acc} \subseteq \theta \times \mathscr{T}$, $\mathit{Eq} \subseteq \theta \times \theta$ and $\mathit{Rev} \subseteq \theta$ are defined by the following mutually recursive rules, where $X, X', Y, Y'$

are variables over $\theta$.

$$T(t) \in \Sigma_{\text{st}} :\!- \mathit{Acc}(T, t) \tag{9}$$

$$\mathit{Eq}(f(\bar{x}), f(\bar{y})) \quad \text{whenever } f(\bar{x}) \in \theta, f(\bar{y}) \in \theta \tag{10}$$

$$\mathit{Rev}(x) \quad \text{whenever } x \in V \tag{11}$$

$$\mathit{Rev}(f(\bar{x})) \quad \text{whenever } f(\bar{x}) \in \theta \tag{12}$$

$$\mathit{Acc}(T, X), \mathit{Triple}(X, p, Y) \in \Sigma_{\text{st}} :\!- \mathit{Acc}(U, Y) \quad \text{for every } \delta(T, p) = U^{\mu} \tag{13}$$

$$\mathit{Acc}(T, X), \mathit{Eq}(X, Y) :\!- \mathit{Acc}(T, Y) \quad \text{for every } T \in \mathscr{T} \tag{14}$$

$$\begin{aligned} &\mathit{Triple}(X, p, Y) \in \Sigma_{\text{st}}, \mathit{Triple}(X', p, Y') \in \Sigma_{\text{st}}, \\ &\quad \mathit{Eq}(X, X'), \\ &\quad \mathit{Acc}(T, X), \mathit{Acc}(T, X') :\!- \mathit{Eq}(Y, Y') \end{aligned} \quad \text{for every } \delta(T, p) = U^{\mu}, \mu \in \{1, ?\} \tag{15}$$

$$\begin{aligned} &\mathit{Triple}(X, p, Y) \in \Sigma_{\text{st}}, \mathit{Triple}(X', p, Y') \in \Sigma_{\text{st}}, \\ &\quad \mathit{Eq}(X, X'), \mathit{Rev}(Y), \\ &\quad \mathit{Acc}(T, X), \mathit{Acc}(T, X') :\!- \mathit{Rev}(Y') \end{aligned} \quad \text{for every } \delta(T, p) = U^{\mu}, \mu \in \{1, ?\} \tag{16}$$

Two terms are *contentious* if they satisfy the relation $\mathit{Cont} \subseteq \theta \times \theta$ defined by

$$\begin{aligned} &\mathit{Triple}(X, p, Y) \in \Sigma_{\text{st}}, \mathit{Triple}(X', p, Y') \in \Sigma_{\text{st}}, \\ &\quad \mathit{Eq}(X, X'), \mathit{Acc}(T, X), \\ &\quad \mathit{Rev}(Y), \mathit{Rev}(Y') :\!- \mathit{Cont}(Y, Y') \end{aligned} \quad \text{for every } \delta(T, p) = U^{\mu}, \mu \in \{1, ?\} \tag{17}$$

This captures the fact that triples generated from the atoms $\mathit{Triple}(X, p, Y)$ and $\mathit{Triple}(X', p, Y')$ might lead to value inconsistency caused by the functional predicate egd for $\delta(T, p) = U^{\mu}$.

The rules defining *Acc*, *Eq*, *Rev* and *Cont* can be turned into a Datalog program over the signature $\{\mathit{Acc}, \mathit{Eq}, \mathit{Rev}, \mathit{Cont}, \_ \in \Sigma_{\text{st}}\}$ by creating as many rules as required by the conditions on $\delta$. The size of $P$ is polynomial in the size of $\mathscr{E}$. Then we can use $P$ to materialize in polynomial time the relations *Acc*, *Eq*, *Rev* and *Cont*.

**Example 31 (Example 14 continued.)** *With the data exchange setting as defined in Example 14, the materialized relations contain the following facts (non exhaustive).*

$$\begin{aligned} &\mathit{Acc}(T, f(x')), \mathit{Acc}(U, g(y'')), \mathit{Acc}(U, g(y)) \\ &\mathit{Rev}(w''), \mathit{Rev}(y') \\ &\mathit{Eq}(g(y''), z'), \mathit{Eq}(g(y), z'), \mathit{Eq}(g(y), g(y'')), \mathit{Eq}(f(x), f(x')) \end{aligned}$$

□

Let $Cont(t, t')$ for some terms $t, t'$ in $\theta$. A *proof tree* for $Cont(t, t')$ is a derivation of the program $P$ which root is $Cont(t, t')$ and which other nodes are facts from the $Acc$, $Eq$, $Rev$ and $\_ \in \Sigma_{\text{st}}$ relations. In particular, the children of the root of such proof tree are $Triple(u, p, t) \in \Sigma_{\text{st}}$, $Triple(u', p, t') \in \Sigma_{\text{st}}$, $Eq(u, u')$, $Acc(T, u)$, $Rev(t)$ and $Rev(t')$ for some terms $u, u'$ and some type $T$ and predicate $p$.

**Example 32 (Example 31 continued.)** *With the data exchange setting as defined in Example 14, there is a proof tree for $Cont(w'', y')$ which nodes are exactly the facts listed in Example 31 and has additionally as leaves all the facts of the form $A \in \Sigma_{\text{st}}$ for all the atoms $A$ that appear in some rule head in $\Sigma_{\text{st}}$:*

$$Triple(f(x), p, g(y)) \in \Sigma_{\text{st}},\ Triple(g(y), q, z) \in \Sigma_{\text{st}}$$
$$T(f(x')) \in \Sigma_{\text{st}},\ Triple(f(x'), p, z') \in \Sigma_{\text{st}},\ Triple(z', q, y') \in \Sigma_{\text{st}}$$
$$Triple(g(y''), q, w'') \in \Sigma_{\text{st}}$$

□

With every such proof tree $\pi$ we can associate an instance $I_{\pi,t,t'}$ of $\mathscr{R}$ s.t. when chased with $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ would produce a violation in which (two constants derived from) the terms $t, t'$ need to be equated by an functional predicate egd for $\delta(T, p)$. The instance $I_{\pi,t,t'}$ is effectively constructed by a *backchase* procedure. We claim that

**Lemma 33** *If $I$ instance of $\mathscr{R}$ is value inconsistent, then there exist two terms $t, t'$ in $\Sigma_{\text{st}}$ s.t. $Cont(t, t')$ holds and there is $I' \subseteq I$ and $\pi$ a proof tree for $Cont(t, t')$ s.t. $I'$ is isomorphic $I_{\pi,t,t'}$.*

The proof is similar the proof of Theorem 8.

Then in order to check value inconsistency of $\mathscr{E}$ it is enough to enumerate all $t, t'$ s.t. $Cont(t, t')$, all proof trees $\pi$ for $Cont(t, t')$ and the corresponding instances $I_{\pi,t,t'}$. If such $I_{\pi,t,t'}$ exists and is a valid instance of $\mathbf{R}$, that is, satisfies the source functional dependencies, then $I_{\pi,t,t'}$ is value inconsistent for $\mathscr{E}$ thus $\mathscr{E}$ is value inconsistent. If there is no $I_{\pi,t,t'}$ that satisfies the source functional dependencies, then $\mathscr{E}$ is value consistent.

**Example 34 (Example 32 continued.)** *With the data exchange setting as defined in Example 14 and the proof tree mentioned in Example 32, we construct the source instance*

$$\{R(x, y, z), S(x, y'), R(x'', y, z'')\}.$$

*One can see that when chasing the above source instance we can derive the facts*

$$\{Triple(f(x), p, g(y)), Triple(g(y), q, \bot_1), U(g(y))$$
$$T(f(x)), Triple(f(x), p, \bot_2), Triple(\bot_2, q, y'), U(\bot_2)$$
$$Triple(g(y), q, w'')\}$$

*Then using the functional predicate egd for $\delta(T, p) = U^1$ we* reveal $\bot_2$ *as being equal to $g(y)$. Finally applying the functional predicate egd for $\delta(U, q) = Literal^?$ we need to equate $w''$ and $y'$ thus the chase fails and $I$ is value inconsistent.* □

On the other hand, node kind consistency of a non-constructive data exchange setting $\mathscr{E}$ can be tested in the same way as for constructive data exchange settings. This concludes the proof of Theorem 15.

## A.3 Complexity of consistency for nondeterministic shape schemas

We reduce the problem of validity of $\forall\exists\mathsf{QBF}$ formulas to testing the consistency of constructive data exchange settings with nondeterministic shape schemas.

We fix a formula $\Phi = \forall\bar{x}.\exists\bar{y}.\varphi$, where $\varphi = c_1 \wedge \ldots \wedge c_k$ is a conjunction of clauses over $\bar{x} = x_1, \ldots, x_n$ and $\bar{y} = y_1, \ldots, y_m$. We construct the following data exchange setting $\mathscr{E}_\Phi$. The relational source schema consists of relations (each relation with a single key)

$$V_{\mathbf{t}}(\underline{x}, y), V_{\mathbf{f}}(\underline{x}, y), R_{x_1}(\underline{x}, y), \ldots, R_{x_n}(\underline{x}, y).$$

We employ a non-overlapping library that for every variable $v \in \bar{x} \cup \bar{y}$ contains the unary IRI constructors $f_v, f_v^{\mathbf{t}}, f_v^{\mathbf{f}}$ and for every clause $c$ it contains a unary $f_c$.

The source-to-target dependencies and the shape schema will introduce a gadget for every variable $v \in \bar{x} \cup \bar{y}$ that will be the only possible source of inconsistency. We identify 3 forms of the gadget, $\mathbf{G}_v$ when the valuation of $v$ is (yet) undetermined, and $\mathbf{G}_v^{\mathbf{t}}$ and $\mathbf{G}_V^{\mathbf{f}}$ for when the variable takes the value true and false respectively. The 3 kinds of gadgets are presented in Figure 3 The

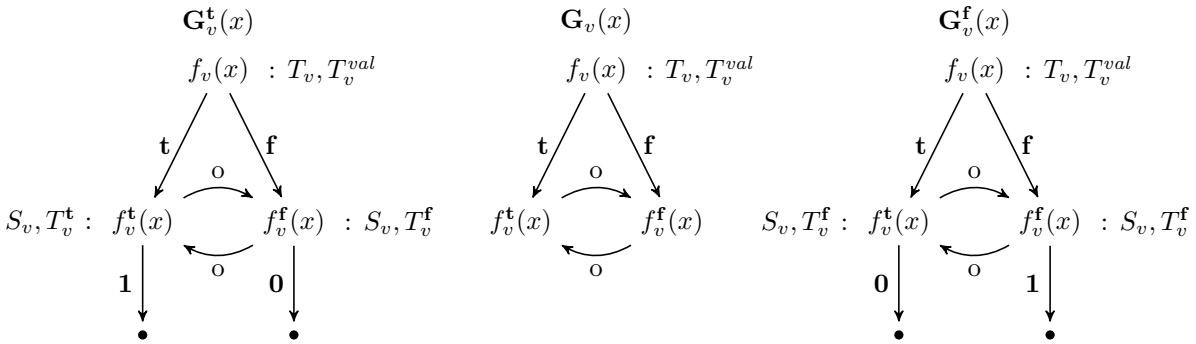


Figure 3: Three gadgets for a variable.

shape schema for every variable $v \in \bar{x} \cup \bar{y}$ contains the following types and their definitions

$$\begin{aligned}
&T_v \to \mathbf{t} :: S_v,\ \mathbf{f} :: S_v\\
&S_v \to o :: S_v, \mathbf{1} :: T_\varnothing?, \mathbf{1} :: T_\varnothing?\\
&T_v^{val} \to \mathbf{t} :: T_v^{\mathbf{t}}?,\ \mathbf{f} :: T_v^{\mathbf{t}}?,\ \mathbf{t} :: T_v^{\mathbf{f}}?,\ \mathbf{f} :: T_v^{\mathbf{f}}?\\
&T_v^{\mathbf{t}} \to \mathbf{1} :: T_\varnothing 1,\ \mathbf{0} :: T_\varnothing 0,\ o :: T_v^{\mathbf{f}}\\
&T_v^{\mathbf{f}} \to \mathbf{1} :: T_\varnothing 0,\ \mathbf{0} :: T_\varnothing 1,\ o :: T_v^{\mathbf{t}}\\
&T_\varnothing \to \epsilon
\end{aligned}$$

We point out that imposing the types $T_v$ and $T_v^{val}$ on a node $f_v(x)$ in $\mathbf{G}_v(x)$ creates a tension that can only be resolved by adding the necessary outgoing

edges **0** and **1** to the nodes $f_v^{\mathbf{t}}(x)$ and $f_v^{\mathbf{f}}(x)$ but only in one of the two ways $\mathbf{G}_v^{\mathbf{t}}(x)$ or $\mathbf{G}_v^{\mathbf{f}}(x)$; in particular no node $f_v^c(x)$ can have both outgoing edges **0** and **1** and every such node must have at least one of the two. In fact, this is the principal reason why an result of chasing any source instance would fail and we shall refer to such situation as type overlap. The schema also contains the following type that shall be used to enforce satisfiability of the clauses

$$\begin{aligned} C &\to l :: T_{\mathbf{t}}\texttt{+}, l :: T_{\mathbf{f}}\texttt{*} \\ T_{\mathbf{t}} &\to \mathbf{1} :: T_{\varnothing}\,\texttt{1} \\ T_{\mathbf{f}} &\to \mathbf{0} :: T_{\varnothing}\,\texttt{1} \end{aligned}$$

Now, the source-to-target dependencies are as follows. For every $x \in \bar{x}$ we have

$$\begin{aligned} V_{\mathbf{t}}(z, y) \wedge R_x(z, y) &\Rightarrow G_x^{\mathbf{t}}(z) \\ V_{\mathbf{f}}(z, y) \wedge R_x(z, y) &\Rightarrow G_x^{\mathbf{f}}(z) \end{aligned}$$

For every $y \in \bar{y}$ we have

$$V_{\mathbf{t}}(x, y_1) \wedge V_{\mathbf{f}}(x, y_2) \Rightarrow G_y(x)$$

For every clause $c$ we also introduce the following st-tgd. Let $c = \ell_1 \vee \ldots \vee \ell_a$, let $v_i$ be the variable used by the literal $\ell_i$, and let $b_i \in \{\mathbf{t}, \mathbf{f}\}$ be the valuation of $v_i$ that satisfies $c$. The st-tgd we introduce for $c$ is

$$V_{\mathbf{t}}(x, y_1) \wedge V_{\mathbf{f}}(x, y_2) \Rightarrow \mathit{Triple}(f_c(x), l, f_{v_1}^{b_1}(x)) \wedge \ldots \wedge \mathit{Triple}(f_c(x), l, f_{v_a}^{b_a}(x)) \wedge C(f_c(x)).$$

For instance, if $c = \neg x_2 \vee y_4 \vee x_3$, then the corresponding st-tgd is

$$\begin{aligned} V_{\mathbf{t}}(x, y_1) \wedge V_{\mathbf{f}}(x, y_2) \Rightarrow\ & \mathit{Triple}(f_c(x), l, f_{x_2}^{\mathbf{f}}(x)) \wedge \\ & \mathit{Triple}(f_c(x), l, f_{y_4}^{\mathbf{t}}(x)) \wedge \\ & \mathit{Triple}(f_c(x), l, f_{x_3}^{\mathbf{t}}(x)) \wedge \\ & C(f_c(x)). \end{aligned}$$

We claim that $\Phi$ is valid if and only if $\mathscr{E}_\Phi$ is consistent.

- For the *if* part we fix an instance $I$ and observe that a result of chasing $I$ may only fail due to type overlap in the nodes of some gadget $G_v(c)$. This is only possible if $c$ is present in both $V_{\mathbf{t}}(c, t)$ and $V_{\mathbf{f}}(c, f)$ for some $t$ and $f$. In fact, for each such $c$ we can consider the subset $I_c \subseteq I$ containing only the facts of $I$ that use $c$ as their key, and the problem can be treated independently for $I_c$. From this subinstance we construct the (possibly partial) valuation $V_C : \bar{x} \to \{\mathbf{t}, \mathbf{f}\}$ and we take any $V^* : \bar{y} \to \{\mathbf{t}, \mathbf{f}\}$ such that $V_c \cup V^* \models \varphi$. We use $V^*$ to construct a consistent solution $J_c$ to $I_c$. The solution to $I$ is obtained from taking the union of all $J_c$'s (and the result of chasing any elements of $I$ that do not belong to any $I_c$ but those cannot create any inconsistency).

- For the *only if* part we take any valuation $V : \bar{x} \to \{\mathbf{t}, \mathbf{f}\}$ and build the instance

$$I_V = \{V_{\mathbf{t}}(c, \mathbf{t}), V_f(c, \mathbf{f})\} \cup \{R_x(c, V(x)) \mid x \in \bar{x}\}$$

Since this is a consistent source instance, it has a solution $J$ for $\mathscr{E}_\Phi$. This solution contains either the gadget $G_y^{\mathbf{t}}(c)$ or $G_y^{\mathbf{f}}(c)$ for every universally quantified variable $y \in \bar{y}$. We construct the corresponding valuation $V^* : \bar{y} \to \{\mathbf{t}, \mathbf{f}\}$. That $V \cup V^* \models \varphi$ follows from the fact that $J$ satisfies the constraints imposed by the type $C$

## A.4 Proofs for Section 6(Certain Query Answering)

Note that $\nleftrightarrow$ is an equivalence relation on nodes of $G$, and we denote by $[n]$ the equivalence class of node $n$ and by $nodes(G)/_{\nleftrightarrow}$ the set of all equivalence classes. For each equivalence class $C \in nodes(G)/_{\nleftrightarrow}$ we fix an arbitrarily chosen representative node $\eta_C \in C$. Now, the *bisimulation quotient* of $G$, denoted by $G/_{\nleftrightarrow}$ is the graph $G/_{\nleftrightarrow} = \{(\eta_{[n]}, p, \eta_{[m]}) \mid (n, p, m) \in G\}$. The choice of the representative does not matter because a non-null value is bisimilar only to itself, and consequently, every non-singleton equivalence class in $nodes(G)/_{\nleftrightarrow}$ contains null values only. The bisimulation quotient of a typed graph $(G, typing)$ is the typed graph $(G/_{\nleftrightarrow}, typing')$, where $typing'(\eta_C) = \bigcup\{typing(n) \mid n \in C\}$ for any $C \in nodes(G)/_{\nleftrightarrow}$. Take a regular acyclic pattern $E$. We claim.

**Lemma 35** *For any two nodes $n$ of $G$ and $m$ of $H$ such that $n \twoheadrightarrow m$, for any $n'$ of $G$, $(n, n') \in [\![E]\!]_G$ implies there is a $m'$ of $H$ such that $(m, m') \in [\![E]\!]_H$ and $n' \twoheadrightarrow m'$*

*Proof.* The proof is by induction on the structure of $E$. The base cases are $E$ with $|E| = 1$ ($E \in \{\epsilon, p, \Box, \langle\ell\rangle\}$). We have the following cases:

- When $E = \epsilon$. Since $n \twoheadrightarrow m$, trivially $(m, m) \in [\![\epsilon]\!]_H$.
- When $E = p$. Assume $(n, n') \in [\![p]\!]_G$. By semantics of $[\![p]\!]_G$ and $n \twoheadrightarrow m$, there is a $m'$ such that $(m, p, m') \in H$. Since $(m, p, m') \in H$, then $(m, m') \in [\![p]\!]_H$ and $n' \twoheadrightarrow m'$
- When $E = \Box$. Assume $(n, n') \in [\![\Box]\!]_G$. By semantics of $[\![\Box]\!]_G$ and $n \twoheadrightarrow m$, there is a $m'$ such that $(m, p, m') \in H$. Since $(m, p, m') \in H$, then $(m, m') \in [\![p]\!]_H$ and $n' \twoheadrightarrow m'$.
- When $E = \langle\ell\rangle$. Assume $(n, n) \in [\![\langle\ell\rangle]\!]_G$. Since $n \twoheadrightarrow m$ and $m \in nodes(G)$ and $[\![\langle\ell\rangle]\!]_G \neq \emptyset$, then $\ell = n = m$. Since $\ell = m$, then $(m, m) \in [\![\langle\ell\rangle]\!]_H$ and $n \twoheadrightarrow m$.

Now assume (IH) that for every expression $E$ with $|E| < i$, we have that for any two nodes $n$ of $G$ and $m$ of $H$ such that $n \twoheadrightarrow m$, for any $n'$ of $G$, if $(n, n') \in [\![E]\!]_G$ then there is a $m'$ such that $(m, m') \in [\![E]\!]_H$ and $n' \twoheadrightarrow m'$. Let $E$ be a regular expression with $|E| = i$. Assume $n \twoheadrightarrow m$. We distinguish the following cases:

1. $E = E_1 + E_2$. We have that $|E_1| < i$ and $|E_2| < i$. Assume $(n, n') \in [\![E]\!]_G$. We have to prove that there is a $m'$ such that $(m, m') \in [\![E]\!]_H$ and $n' \twoheadrightarrow m'$. Let $m'$ be in $H$. By definition, $(n, n') \in [\![E_1]\!]_G \cup [\![E_2]\!]_G$. By IH, there is $m_1$ of $H$ such that $(m, m_1) \in [\![E_1]\!]_G$ and $n' \twoheadrightarrow m_1$. Let $m_1 = m'$. By IH, there is $m_2$ of $H$ such that $(m, m_2) \in [\![E_2]\!]_G$ and $n' \twoheadrightarrow m_2$. Let $m_2 = m'$. By $(m, m_1) \in [\![E_1]\!]_G$ and $(m, m_2) \in [\![E_2]\!]_G$ and $m_1 = m = m_2$, we have that $(m, m') \in [\![E_1]\!]_H \cup [\![E_2]\!]_H$. Hence, we conclude $(m, m') \in [\![E]\!]_H$ and $n' \twoheadrightarrow m'$.

2. $E = E_1 \cdot E_2$. We have that $|E_1| < i$ and $|E_2| < i$. Assume $(n, n') \in [\![E]\!]_G$. By semantics of $\cdot$, $(n, n') \in [\![E_1]\!]_G \circ [\![E_2]\!]_G$. By composition of binary relations, there is $n_2$ such that $(n, n_2) \in [\![E_1]\!]_G$ and $(n_2, n') \in [\![E_2]\!]_G$. Since $(n, n_2) \in [\![E_1]\!]_G$ and $n \twoheadrightarrow m$, then there is $m_2$ such that $(m, m_2) \in [\![E_1]\!]_H$. By applying IH, we have that $n_2 \twoheadrightarrow m_2$, and together with $(n_2, n') \in [\![E_2]\!]_G$, we obtain that $(m_2, m') \in [\![E_2]\!]_H$ for some $m'$ in $H$. We can use IH to conclude that $n' \twoheadrightarrow m'$. By $(m, m_2) \in [\![E_1]\!]_H$ and $(m_2, m') \in [\![E_2]\!]_H$, we have that $(m, m') \in [\![E_1]\!]_H \circ [\![E_2]\!]_H$. By semantics of $\cdot$, $(m, m') \in [\![E_1 \cdot E_2]\!]_H$. Thus, we obtain $(m, m') \in [\![E]\!]_H$ and $n' \twoheadrightarrow m'$.

3. $E = E'^*$. We have that $|E'| = i - 1$. Assume $(n, n') \in [\![E]\!]_G$. Since $i > 1$ and by semantics of $*$, we have that $(n, n') \in \bigcup_{2 \leq k} [\![E']\!]^k_G$. Applying transitive closure of binary relation, we obtain $(n, n') \in [\![E']\!]^2_G \cup [\![E'^*]\!]_G$ and $|E'^*| < i$. Then, we have that $(n, n') \in [\![E']\!]_G \circ [\![E']\!]_G$, or $(n, n') \in [\![E'^*]\!]_G$. By $n \twoheadrightarrow m$ and the prove of $E_1 \cdot E_2$, we know that there is $(m, m') \in [\![E']\!]_G \circ [\![E']\!]_H$ and $n' \twoheadrightarrow m'$. Applying IH in $|E'^*| < i$, there is $(m, m') \in [\![E'^*]\!]_H$ and $n' \twoheadrightarrow m'$. By the two statements above, we have $(m, m') \in \bigcup_{2 \leq k} [\![E']\!]^k_H$ and $n' \twoheadrightarrow m'$. Thus, we conclude $(m, m') \in [\![E]\!]_H$ and $n' \twoheadrightarrow m'$.

4. $E = [E']$. Assume $(n, n) \in [\![[E]]\!]_G$ and $n \twoheadrightarrow m$. By definition of $[E']$, there is $n'$ such that $(n, n') \in [\![E']\!]_G$. Applying IH, there is $m'$ such that $(m, m') \in [\![E']\!]_H$. By definition, $(m, m) \in [\![[E']]\!]_H$ and $n \twoheadrightarrow m$. Thus, we conclude that $(m, m) \in [\![E]\!]_H$.

Finally, we claim.

**Lemma 36** *For any two graphs $G$ and $H$, if $G \twoheadrightarrow H$ then if $G \models E$ implies $H \models E$.*

*Proof.* Take any two graphs $G$ and $H$. Assume $G \twoheadrightarrow H$ and $G \models E$. By definition of $G \models E$, $[\![E]\!]_G \neq \emptyset$, and in consequence there is a pair $(n, n') \in [\![E]\!]_G$. By $G \twoheadrightarrow H$, there is a node $m$ in $H$ such that $n \twoheadrightarrow m$. By lemma 35 and $n \twoheadrightarrow m$ and $(n, n') \in [\![E]\!]_G$, there is $m'$ of $H$ such that $(m, m') \in [\![E]\!]_H$. Since $[\![E]\!]_H \neq \emptyset$, then $H \models E$.

Take two graphs $G$ and $H$. Assume $G \models E$. By lemma 36, it holds that $H \models E$. Consequently, $E$ is robust under simulation.

### A.4.1 Proof of Theorem 20

Take any consistent instance $I$ of $\mathbf{R}$. Now, we prove that the typed graph $\mathscr{U} = J_0 \cup G_{\mathbf{S}}$, is a universal simulation solution. First, we define *reachability* from a node $n$ with a path $\pi$ as follows:

$$\begin{aligned} \mathcal{R}_G(N, p) &= \{n' \mid \exists n \in N.\ \mathit{Triple}(n, p, n') \in G\} \\ \mathcal{R}^*_G(n, \pi \cdot p) &= \mathcal{R}_G(\mathcal{R}^*_G(n, \pi), p) \\ \mathcal{R}^*_G(n, \epsilon) &= \{n\} \end{aligned}$$

Then, we extend the canonical function $\Delta$ as follows.

- $\Delta^*(X, \pi \cdot p) = \Delta(\Delta^*(X, \pi), p)$

- $\Delta^*(X, \epsilon) = X$

Now, we claim

**Lemma 37** *For any solution $J$ to $\mathscr{E}$ for $I$ and for any frontier $(n_0, p_0) \in \mathbb{F}$ and for any path in $\mathscr{U}$ of the form $\pi = p_0 \cdot p_1 \cdot \ldots \cdot p_k$. Let $X = \Delta^*(types_{J_0}(n_0), \pi)$. Then*

1. *$\pi$ is also in $J$*
2. *if $n = \mathcal{R}^*_{\mathscr{U}}(n_0, \pi)$ then $types_{\mathscr{U}}(n) = X$*
3. *$\forall m \in \mathcal{R}^*_J(n_0, \pi).\, X \subseteq types_J(m)$*

*Proof.* Take any solution $J$ for $I$ to $\mathscr{E}$ and any frontier $(n_0, p_0) \in \mathbb{F}$. We prove by induction in the size of the path $\pi$. Let $|\pi| \leq k$. The base case is when $\pi$ is of size 1, i.e. $\pi = p_0$. Let $X = \Delta^*(types(n_0), \pi)$. Then

1. For case 1. We know $(n_0, p_0) \in \mathbb{F}$ and because $p_0$ is in $\mathscr{U}$ then $p_0 \in Req(types_{J_0}(n_0))$. Since $J$ is a solution then the IRIs required by $types_{J_0}(n_0)$ must be satisfied. Since $p_0 \in Req(types_{J_0}(n_0))$ then there is $m$ such that $Triple(n_0, p_0, m) \in J$. Thus, $\pi$ is in $J$.

2. For case 2. Assume $n = \mathcal{R}^*_{\mathscr{U}}(n_0, p_0)$. By definition of $\Delta^*$, we have $X = \Delta(types_{J_0}(n_0), p_0)$. Then, we take a type $T \in types_{\mathscr{U}}(n)$, and by definition, we have $T(n) \in G_{\mathbf{S}}$, $n \in N$ and $T \in n$. By construction of $G_{\mathbf{S}}$ and $(n_0, p_0) \in \mathbb{F}$, we obtain $n \in N_0$. Also by construction of $N_0$ and $X$ definition, we obtain $n = \Delta(types_{J_0}(n_0), p_0) = X$. Finally, by construction of $G_{\mathbf{S}}$, we have that $T \in X$. Similar process is done in left direction. Thus, we conclude that $types_{\mathscr{U}}(n) = X$.

3. For case 3. Take any $m \in \mathcal{R}_J(n_0, p_0)$, i.e. $Triple(n_0, p_0, m) \in J$. By definition of $\Delta^*$, we have $X = \Delta(types_{J_0}(n_0), p_0)$. Then, take any $T \in X$. Since $(n_0, p_0) \in \mathbb{F}$ and $J$ is a solution, then there is a type $T'$ such that $T'(m) \in J$. Let $T' = T$. By $T(m) \in J$, we have that $T \in types_J(m)$. Thus, we conclude that $X \subseteq types_J(m)$.

Now we fix $k > 1$ and assume (IH) for any path $\pi$ in $\mathscr{U}$ such that $|\pi| \leq k$ holds that (a) $\pi$ is also valid in $J$, and (b) if $n = \mathcal{R}_{\mathscr{U}}(n_0, \pi)$ then $types_{\mathscr{U}}(n) = X$; and (c) for any $m \in \mathcal{R}_J(n_0, \pi)$ holds that $X \subseteq types_J(m)$.

Let $|\pi| \leq k + 1$. Take any path $\pi$ in $\mathscr{U}$ such that $|\pi| \leq k + 1$. Let $X = \Delta^*(types(n_0), \pi)$ and $\pi = \pi' \cdot p$ such that $|\pi'| \leq k$. We have the following cases:

1. Case 1. By definition of path, there is $n_0, \ldots, n_{k+1}$ such that $(n_{i-1}, p_i, n_i) \in \mathscr{U}$ for $i \in \{1 \ldots, k, k+1\}$ and $\pi' = p_1 \cdot \ldots \cdot p_k$. Applying IH, we have that $\pi'$ is a path in $J$ and there are $m, m_k$ in $J$ such that $Triple(m, p_k, m_k) \in J$. Since $(n_0, p_0) \in \mathbb{F}$ and $p_{k+1} \in Req(X)$ and $J$ is a solution, then it holds that $Triple(m_k, p_{k+1}, m_{k+1}) \in J$. Thus, $\pi$ is a path in $J$.

2. Case 2. Assume $n = \mathcal{R}^*_{\mathscr{U}}(n_0, \pi' \cdot p)$. Let $X' = \Delta^*(types_{J_0}(n_0))$. By definition of $\Delta^*$, it is equivalent to $X = \Delta(X', p)$. Let $n_1 = \mathcal{R}^*_{\mathscr{U}}(n_0, \pi')$. By definition of $\mathcal{R}^*_{\mathscr{U}}$, we have $\mathcal{R}_{\mathscr{U}}(n_1, p) = n$. Applying IH, we obtain that $types_{\mathscr{U}}(n_1) = X'$, and by definition of path $\pi' \cdot p$, we have that

*Triple*$(n_1, p, n) \in \mathscr{U}$. Now, we take any $T \in X$. By the statements above and considering that $p \in \mathit{Req}(X')$ and by construction of $\mathscr{U}$, we have that $T(n) \in \mathscr{U}$, i.e. $T \in \mathit{types}_{\mathscr{U}}(n)$. A similar process is done for proving the right direction. Thus, we conclude that $\mathit{types}_{\mathscr{U}}(n) = X$.

3. Case 3. Take $m' \in \mathcal{R}^*_J(n_0, \pi' \cdot p)$. By definition of $\mathcal{R}_J$, there is *Triple*$(m, p, m') \in J$ where $m \in \mathcal{R}^*_J(n_0, \pi')$. Now, we take any $T \in X$ and applying the IH, we obtain that there is a type $T'$ such that $T'(m) \in J$. Since $J$ is a solution and $p \in \mathit{Req}(X)$ and statements above, we have that $T(m') \in J$, i.e. $T \in \mathit{types}_J(m')$. Thus, we conclude that $X \subseteq \mathit{types}_J(m)$.

Next, we claim the following.

**Lemma 38** *$\mathscr{U}$ is simulated by every solution $J$ for $I$ to $\mathscr{E}$.*

*Proof.* We construct

$$R = \{(n, m) \in J_0 \times J_0 \mid n = m\} \cup$$
$$\{(n, m) \mid \exists (n_0, p_0) \in \mathbb{F}.\, \exists \pi = p_0 \cdot p_1 \cdot \ldots \cdot p_k.\, n \in \mathcal{R}_{\mathscr{U}}(n_0, \pi) \wedge m \in \mathcal{R}_J(n_0, \pi)\}.$$

We show that $R$ is a simulation of $\mathscr{U}$ by $J$. Then, we take any pair $(n, m) \in R$ and $p \in \mathsf{Iri}$. We have the following cases:(a) $n \in J_0 \wedge (n, p) \notin \mathbb{F}$ and (b) $(n, p) \in \mathbb{F} \vee n \in \mathit{nodes}(G_{\mathbf{S}})$.

For case a. We know that $(n, m) \in J_0 \times J_0$ and $n = m$. We take $n' \in \mathit{nodes}(J_0)$ such that *Triple*$(n, p, n') \in J_0$. As a result of consider $m' = n'$, we obtain $m' \in \mathit{nodes}(J_0)$ then $m' \in \mathit{nodes}(J)$. Since *Triple*$(n, p, n') \in J_0$, we have *Triple*$(m, p, m') \in J_0$, and by $(n', m') \in J_0 \times J_0$, we conclude $(n', m') \in R$.

For case b. We prove only when $n \in \mathit{nodes}(G_{\mathbf{S}})$ since the other is implied by this proof. By $n \in \mathit{nodes}(G_{\mathbf{S}})$ and $(n, m) \in R$, we have that $n = \mathcal{R}^*_{\mathscr{U}}(n_0, \pi)$ where $\pi = p_1 \cdot \ldots \cdot p_k$ and $(n_0, p_1) \in \mathbb{F}$ and $m \in \mathcal{R}_J(n_0, \pi)$. Then, we take $p, n'$ such that *Triple*$(n, p, n') \in \mathscr{U}$, i.e. $n' \in \mathcal{R}^*_{\mathscr{U}}(n_0, \pi \cdot p)$ and $\pi \cdot p$ is valid in $\mathscr{U}$. By lemma 37, we have $\pi \cdot p$ is a path in $J$, i.e. there is a node $m' \in \mathcal{R}_J(n_0, \pi \cdot p)$. Thus, we conclude that $(n', m') \in R$. By lemma 38, $\mathscr{U}$ is a universal simulation solution. Then the proof is relatively straightforward, and for the *if* part, it suffices to use Lemma 18 and for the *only if* part, it suffices to notice that a universal simulation solution is also a solution.

### A.4.2 Proof of Theorem 21

Before proving this theorem, we show that $\mathscr{U}_0$ is indeed the minimal universal simulation solution. We take any universal simulation solution $\mathscr{U}$ and create an injective mapping from the nodes of $\mathscr{U}_0$ to the nodes of $\mathscr{U}$. The mapping is an identity on $J_0$ which is contained in any solution. Now, for a node $n$ of $G_{\mathbf{S}}/\!\leftrightarrow$ we observe that there must be at least one path $\pi$ from a frontier node $n_0$ to $n$, and because $\mathscr{U}$ is simulated in $\mathscr{U}_0$, there exists at least one node $m$ in $\mathscr{U}$ that is reachable from $n_0$ by path $\pi$. Consequently, we map $n$ to an arbitrary such $m$. Now, suppose that two different nodes $n_1$ and $n_2$ of $G_{\mathbf{S}}/\!\leftrightarrow$ are mapped to the same node $m$. Because $\mathscr{U}_0$ is a bisimulation quotient and the nodes $n_1$ and $n_2$ are different, they are not bisimilar. However, since $\mathscr{U}_0$ is simulated by $\mathscr{U}$, and vice versa, and $n_1$ is reachable with the same path in $\mathscr{U}_0$ as $m$ in $\mathscr{U}$ and $n_1$ is reachable with the same path in $\mathscr{U}_0$ as $m$ in $\mathscr{U}$, $n_1$ is bisimilar to $m$

and $m$ is bisimilar to $n_2$. By transitivity of bisimulation, we get that $n_1 \nleftrightarrow\!\!\!\!\!\rightarrow n_2$, a contradiction.

Take an instance $I$ of $\mathbf{R}$. We construct a typed graph as follows $\mathscr{U}_0 = J_0 \cup G_{\mathbf{S}}/_{\nleftrightarrow\!\!\!\!\!\rightarrow}$ where $G_{\mathbf{S}}/_{\nleftrightarrow\!\!\!\!\!\rightarrow}$ is the bisimulation quotient of $G_{\mathbf{S}}$. We claim that

**Lemma 39** *$\mathscr{U}_0$ is a universal simulation solution.*

*Proof.* The proof is similar to Lemma 38. We construct a relation as follows:

$$R = \{(n,m) \in J_0 \times J_0 \mid n = m\} \cup \{(C,m) \mid \exists (n_0,p_0) \in \mathbb{F}.\, \exists \pi = p_0 \cdot p_1 \cdot \ldots \cdot p_k. \\ \exists n \in C.\, n \in \mathcal{R}_{\mathscr{U}}(n_0,\pi) \wedge m \in \mathcal{R}_J(n_0,\pi)\}.$$

The first case is the same as lemma 38. The second case is proven when $C \in \mathit{nodes}(G_{\mathbf{S}})/_{\nleftrightarrow\!\!\!\!\!\rightarrow}$. Since $C \in \mathit{nodes}(G_{\mathbf{S}})/_{\nleftrightarrow\!\!\!\!\!\rightarrow}$ and $(C,m) \in R$, then there is $n \in C$ such that $n = \mathcal{R}^*_{\mathscr{U}}(n_0,\pi)$ where $\pi = p_1 \cdot \ldots \cdot p_k$ and $(n_0,p_1) \in \mathbb{F}$ and $m \in \mathcal{R}_J(n_0,\pi)$. Now, we take $p \in \mathsf{Iri}, C' \in \mathit{nodes}(G_{\mathbf{S}})/_{\nleftrightarrow\!\!\!\!\!\rightarrow}$ such that $(\eta_C, p, \eta'_C) \in \mathscr{U}_0$, i.e., there are $n \in C$ and $n' \in C'$ such that $(n,p,n') \in G_{\mathbf{S}}$. From this fact, we have that $n' \in \mathcal{R}^*_{\mathscr{U}_0}(n_0, \pi \cdot p)$ and $\pi \cdot p$ is valid in $\mathscr{U}_0$. By lemma 37, $\pi \cdot p$ is valid in $J$, i.e., there is a node $m' \in \mathcal{R}_J(n_0, \pi \cdot p)$. As a consequence, we conclude that $(C', m') \in R$ yielding that $\mathscr{U}_0$ is a universal simulation solution. Next, we claim.

**Lemma 40** *For any universal simulation solution $\mathscr{U}$ it holds, $|\mathscr{U}| \geq |\mathscr{U}_0|$*

*Proof.* We take any universal simulation solution $\mathscr{U}$ and create an injective mapping from the nodes of $\mathscr{U}_0$ to the nodes of $\mathscr{U}$. The mapping is an identity on $J_0$ which is contained in any universal simulation solution. Now for a node $n$ of $G_{\mathbf{S}}/_{\nleftrightarrow}$ we observe that there must be at least one path $\pi$ from a frontier node $n_0$ to $n$, and because $\mathscr{U}$ is simulated in $\mathscr{U}_0$, there exists at least one node $m$ in $\mathscr{U}$ that is reachable from $n_0$ by path $\pi$. Consequently, we map $n$ to an arbitrary such $m$. Now, suppose that two different nodes $n_1$ and $n_2$ of $G_{\mathbf{S}}/_{\nleftrightarrow}$ are mapped to the same node $m$. Because $\mathscr{U}_0$ is a bisimulation quotient and the nodes $n_1$ and $n_2$ are different, they are not bisimilar. However, since $\mathscr{U}_0$ is simulated by $\mathscr{U}$, and vice versa, and $n_1$ is reachable is reachable with the same path in $\mathscr{U}_0$ as $m$ in $\mathscr{U}$ and $n_1$ is reachable with the same path in $\mathscr{U}_0$ as $m$ in $\mathscr{U}$, $n_1$ is bisimilar to $m$ and $m$ is bisimilar to $n_2$. By transitivity of bisimulation, we get that $n_1 \nleftrightarrow\!\!\!\!\!\rightarrow n_2$, a contradiction. Finally, we claim.

**Lemma 41** *There is a polynomial formula such that for any $n, m \in \mathbb{N}$, there exists a data exchange setting $\mathscr{E}$ and instance $I$ of $\mathbf{R}$ such that the size of $\mathscr{U}_0$ is asymptotic to $\exp(m)$ and it holds $|\mathscr{E}| + |I| \leq \mathit{poly}(n)$ where $\mathit{poly}(n)$ is the polynomial formula.*

*Proof.* Let $I = \{R(1)\}$ and $\Sigma_{\mathrm{st}}$ contains only $R(x) \Rightarrow T(f(x))$ and $\mathbf{S}$ be as in Figure 4 that contain cycles of length $2, 3, 5, \ldots$, prime numbers with one shape type name different such as $T_{23}, T_{34}$, and $T_{56}$. Let $P_m$ stands for the $m$-th prime number for $m \in \mathbb{N}$. When constructing the universal simulation solution $\mathscr{U}_0$ we can observe that $|\mathscr{U}_0| \equiv 1(\mod 2)$ and $|\mathscr{U}_0| \equiv 1(\mod 3)$ and so on. Then, we can apply the chinese reminder theorem such that $|\mathscr{U}_0| \equiv 1(\mod k)$ such that $k = 2 * 3 \ldots * P_m$. The product of $m$ prime numbers is approximately $2 * 3 * \ldots * P_m \leq 2^{2m}$. We compute the size of the universal simulation solution

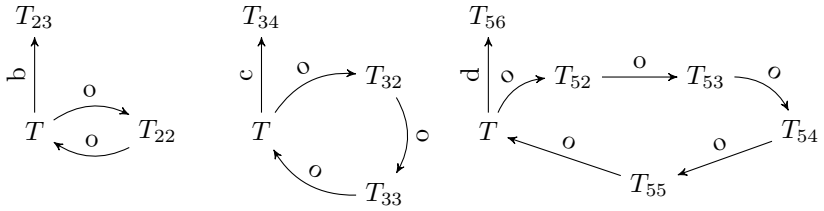


Figure 4: Shape Schema Graph

$\mathscr{U}$ using the prime number counting function, denoted by $\pi(m)$ that counts the number of primes less or equal to $m \in \mathbb{N}$. It follows from the prime number theorem that for all $m \in \mathbb{N}, \pi(m) \sim m/\log(m)$, i.e., $\lim_{m\to\infty}(\pi(m) \times \log(m)/m) = 1$. The prime number theorem guarantees that the set of all natural numbers up to a fixed size asymptotically contains an exponential number of prime number. By the prime number theorem, $P_m$ is asymptotic to $m * \log m$ as $m \to \infty$. The sum of $m$ prime numbers is

$$\begin{aligned} 2 + 3 + \ldots + P_m &\leq m * P_m \\ &\leq m * m * \log m \\ &\leq m^3 \end{aligned}$$

Let $n = |I|$. Since the application of $\Sigma_{\text{st}}$ is founding an homomorphism in every tuple of $I$, in the worst case we can have that the size of the core pre-solution $|J_0| \leq n^2$. Let $poly(n) = n^2 + 1$. Finally, we get for $m, n \in \mathbb{N}$, the size of $|\mathscr{U}_0| \leq n^2 + 2^{2m/3}$. Let $\exp(m) = 2^{2m/3}$. Thus, $\mathscr{U}_0$ is asymptotic to $\exp(m)$ and $|\mathscr{E}| + |I| \leq poly(n)$. By lemma 41, the size of $\mathscr{U}_0$ is bounded by a polynomial in the size of $I$ and an exponential function in the size of $\mathbf{S}$. Since $\mathscr{U}_0$ exists, then we can construct a size-minimal universal simulation solution.

### A.4.3 Proof of Theorem 22

Let $\mathscr{E}$ be a constructive data exchange setting with fixed IRI constructors, and $Q$ a regular acyclic pattern. Take any instance $I$ of $\mathbf{R}$.

Following the semantics of nSPARQL [24], we find equivalences to NRE$^{\rightarrow}$ as follows:

$$\begin{aligned} [\![\epsilon]\!]_G &= [\![\mathsf{self}]\!]_G, & [\![[E]]\!]_G &= [\![\mathsf{self} :: [exp]]\!]_G, \\ [\![p]\!]_G &= [\![\mathsf{next} :: a]\!]_G, & [\![E_1 + E_2]\!]_G &= [\![exp_1 | exp_2]\!]_G, \\ [\![\Box]\!]_G &= [\![\mathsf{next}]\!]_G, & [\![E_1 \cdot E_2]\!]_G &= [\![exp_1 / exp_2]\!]_G, \\ [\![\langle \ell \rangle]\!]_G &= [\![\mathsf{self} :: a]\!]_G, & [\![E^*]\!]_G &= [\![exp^*]\!]_G, \end{aligned}$$

where $\mathsf{next}, \mathsf{self}$ are navigational axes that nSPARQL uses and $exp$ is an expression in nSPARQL where the axis of expression $\in \{\mathsf{next}, \mathsf{self}\}$.

We use the polynomial decision algorithm presented in section 3.1 of [24] for evaluating $Q$ in $I$ w.r.t. $\mathscr{E}$. It is known in [24] that the evaluation of a query in a graph is $O(|G| \cdot |exp|)$. By theorem 21, we construct a size-minimal universal simulation solution for $I$ to $\mathscr{E}$ such that its size bounded by a polynomial in the size of $I$ and an exponential function in the size of $\mathbf{S}$. Since $\mathbf{S}$ is fixed and

because the universal simulation solution is bounded, the data complexity is $O(|\mathscr{U}_0| \cdot |E|)$, which is $O((n^2 + 2^{c*m}) \cdot |E|)$ where $n$ is the size of $I$ and $m$ is the number of shape types.

## A.5 Proof of Proposition 23

The proof is by reduction of intersection non-emptiness of $n$ regular expressions $E_1, \ldots, E_n$ over $\Sigma$. Indeed, we only need a simple schema $\delta(T, a) = T^+$ for $a \in \Sigma$, a single st-tgd $R(x) \Rightarrow T(f(x))$, and a instance $I = \{R(0)\}$. If we let $\# = f(0)$, then *true* is the consistent answer to $Q = \# \cdot \Box^* \cdot [E_1^- \cdot \#^-] \cdot \ldots \cdot [E_n^- \cdot \#^-]$ if and only if $E_1 \cap \ldots \cap E_n$ is nonempty.