Data Exchange from Relational Databases to RDF with Target Shape Schemas

Jose Martin Lozano Aparicio

Université de Lille

Centre de Recherche en Informatique, Signal et Automatique de Lille
Region Hauts-de-France

THÈSE

présentée et soutenue le 14 décembre 2020

Spécialité : INFORMATIQUE

par

Jose Martin LOZANO APARICIO

Data Exchange from Relational Databases to RDF with Target Shape Schemas

Échange de données de bases de données relationnelles vers RDF avec des schémas des constraintes sur cible

sous la direction de

Sławek STAWORKO

et

Iovka BONEVA

Jury :

Rapporteurs :	Jef WIJSEN	-	Université de Mons
	Mirian HALFELD-FERRARI	-	Université d'Orléans
Directeur :	Sławek STAWORKO	-	Université de Lille
Co-encadrante :	Iovka BONEVA	-	Université de Lille
Examinateurs :	Anne ETIEN	-	Université de Lille
	Federico ULLIANA	-	Université de Montpellier

Abstract: Resource Description Framework (RDF) is a graph data model which has recently found the use of publishing on the web data from relational databases. We investigate data exchange from relational databases to RDF graphs with target shapes schemas. Essentially, *data exchange* models a process of transforming an instance of a relational schema, called the *source schema*, to a RDF graph constrained by a *target schema*, according to a set of rules, called *source-to-target tuple generating dependencies*. The output RDF graph is called a *solution*. Because the tuple generating dependencies define this process in a declarative fashion, there might be many possible solutions or no solution at all. We study *constructive* relational to RDF data exchange setting with target shapes schemas, which is composed of a relational source schema, a shapes schema for the target schema, a set of mappings that uses IRI constructors. Furthermore, we assume that any two IRI constructors are non-overlapping.

We propose a *visual mapping language* (VML) that helps non-expert users to specify mappings in this setting. Moreover, we develop a tool called ShERML that performs data exchange with the use of VML and for users that want to understand the model behind VML mappings, we define R2VML, a text-based mapping language, that captures VML and presents a succinct syntax for defining mappings.

We investigate the problem of checking consistency: a data exchange setting is *consistent* if for every input source instance, there is at least one solution. We show that the consistency problem is coNP-complete and provide a static analysis algorithm of the setting that allows to decide if the setting is consistent or not.

We study the problem of computing *certain answers*. An answer is *certain* if the answer holds in every solution. Typically, certain answers are computed using a universal solution. However, in our setting a universal solution might not exist. Thus, we introduce the notion of *universal simulation solution*, which always exists and allows to compute certain answers to any class of queries that is *robust under simulation*. One such class is 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 investigate the shapes *schema elicitation problem* that consists of constructing a target shapes schema from a constructive relational to RDF data exchange setting without the target shapes schema. We identity two desirable properties of a good

target schema, which are *soundness* i.e., every produced RDF graph is accepted by the target schema; and *completeness* i.e., every RDF graph accepted by the target schema can be produced. We propose an elicitation algorithm that is sound for any schema-less data exchange setting, but also that is complete for a large practical class of schema-less settings.

Keywords: Data Exchange, Shapes Schema, Certain Query Answering, Consistency, Visual Mapping Language, Schema Elicitation

Résumé: Resource Description Framework (RDF) est un modèle de graphe utilisé pour publier des données sur le Web à partir de bases de données relationnelles. Nous étudions l'échange de données depuis des bases de données relationnelles vers des graphes RDF avec des schémas de formes cibles. Essentiellement, *échange de données* modélise un processus de transformation d'une instance d'un schéma relationnel, appelé *schéma source*, en un graphe RDF contraint par un *schéma cible*, selon un ensemble de règles, appelé *tuple source-cible générant des dépendances*. Le graphe RDF obtenu est appelé une *solution*. Étant donné que les dépendances générant des tuple définissent ce processus de manière déclarative, il peut y avoir de nombreuses solutions possibles ou aucune solution du tout. Nous étudions le système d'échange de données relationnel avec RDF *constructive* avec des schémas de formes cibles, qui est composé d'un schéma source relationnel, un schéma de formes pour le schéma cible, un ensemble de mappages utilisant des constructeurs IRI. De plus, nous supposons que deux constructeurs IRI ne se chevauchent pas.

Nous proposons un *langage visuel pour l'spécification des correspondances* (VML) qui aide les utilisateurs non experts à spécifier des mappages dans ce système. De plus, nous développons un outil appelé ShERML qui effectue l'échange de données avec l'utilisation de VML et pour les utilisateurs qui souhaitent comprendre le modèle derrière les mappages VML, nous définissons R2VML, un langage texte, qui capture VML et présente une syntaxe succincte pour définition des mappages.

Nous étudions le problème de la vérification de la consistance: un système d'échange de données est *consistent* si pour chaque instance de source d'entrée, il existe au moins une solution. Nous montrons que le problème de consistance est coNP-complet et fournissons un algorithme d'analyse statique du système qui permet de décider si le système est consistent ou non.

Nous étudions le problème du calcul de *réponses certaines*. Une réponse est *certain* si la réponse tient dans chaque solution. En générale, réponses certaines sont calculées en utilisant d'une solution universelle. Cependant, dans notre contexte, une solution universelle pourrait ne pas exister. Ainsi, nous introduisons la notion de *solution de simulation universelle*, qui existe toujours et permet de calculer certaines réponses à n'importe quelle classe de requêtes *robustes sous simulation*. Une de ces classes sont les expressions régulières imbriquées (NRE) qui sont *forward* c'est-à-dire qui n'utilisent

pas le opération inverse. L'utilisation d'une solution de simulation universelle rend traitable le calcul de réponses certaines pour les NRE (data-complexity).

Enfin, nous étudions le *problème d'extraction de schéma des formes* qui consiste à construire un schéma de formes cibles à partir d'un système constructif d'échange de données relationnel vers RDF sans le schéma de formes cibles. Nous identifions deux propriétés souhaitables d'un bon schéma cible, qui sont la *correction* c'est-à-dire que chaque graphe RDF produit est accepté par le schéma cible; et la *complétude* c'est-à-dire que chaque graphe RDF accepté par le schéma cible peut être produit. Nous proposons un algorithme d'extraction qui convient à tout système d'échange de données sans schéma, mais qui est également complet pour une grande classe pratique de systèmes sans schéma.

Mots-clés: Échange de données, schéma de formes, calcul des réponses certaines, consistance, langage visuel pour la spécification des correspondances, extraction de schéma

Acknowledgements

First and foremost, I am grateful to the almighty God for blessing and guiding me in all the works performed for the completion of this research thesis work with his presence to complete successfully. Each moment during the course of this work, I experienced the grace of our Lord Jesus Christ, who continuously enhanced my intelligence even at the moments of despair, inspired me to move forward, opened before me unexpected avenues and enlightened my thoughts with His wisdom. Also, I am grateful to the Blessed Virgin Mary and all saints in heaven who with their intercession I obtained the perseverance to finish this thesis.

I would like to express gratitude to my supervisor Dr. Sławek Staworko and co-advisor Dr. Iovka Boneva. I appreciate their simplicity and art of feeding knowledge patiently to bring out the researching capability and deliver my research product successfully all through the endeavour.

This research benefits from the support of the *Région Hauts-de-France* by a grant from CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020 and by the ANR project DataCert ANR-15-CE39-0009.

Last but not least, I would be glad to thank my dear parents, Nilda and Walter, and my dear wife, Edith. My immense gratitude to them, for their encouragement and advice. Their support throughout the years has been unwavering.

Contents

Introduction

Motivation

Resource Description Framework (RDF) [Lassila & Swick 1999] is a well-established format that finds usage as data exchange language between web applications [Baker *et al.* 2012]. In many of these web applications, the data is exported from relational databases as evidenced by the proliferation of languages for mapping relational databases to RDF, such as R2RML [Das *et al.* 2011], direct mapping (DM) [Arenas *et al.* 2012] or YARRRML [Heyvaert *et al.* 2018] .

Take for instance R2RML [Das *et al.* 2011], which is a declarative mapping language recommended by W3C that allows the customization of what information from a relational database is relevant to be exported to RDF. As an example, consider the two following R2RML mappings, themselves in RDF format presented in Turtle syntax, exporting data about employees and departments from a relational database to RDF.

```
<#DeptMap>
  rr:logicalTable[ rr:tableName “Dept”];
  rr:subjectMap[rr:template “dept:{did}”; rr:class :TDept];
  rr:predicateObjectMap[rr:predicate :name; rr:objectMap[rr:column “name”]].
<#EmpMap>
  rr:logicalTable[ rr:sqlQuery “SELECT eid, name, email, did
                                FROM Emp NATURAL JOIN Email ”];
  rr:subjectMap[rr:template “emp:{eid}”; rr:class :TEmp];
  rr:predicateObjectMap[rr:predicate :name; rr:objectMap[rr:column “name”]];
  rr:predicateObjectMap[rr:predicate :email; rr:objectMap [ rr:column “email”]];
  rr:predicateObjectMap [rr:predicate :works; rr:objectMap[rr:template “dept:{did}”]];
```

In general, R2RML maps data from relational tables or query outputs and produces set of triples using templates for subjects, explicitly stated predicate names, and for objects either literal values coming from columns or templates. Additionally, a type for the subject can be declared. For instance, the first mapping maps data from relation $Dept(\underline{did}, name)$ into a set of triples with predicate :name. For every department, it creates a dedicated Internationalized Resource Identifier (IRI) and the class (rr:class) of each department IRI is declared as :TDept.

RDF has been originally proposed schema-less to promote its adoption but the need for schema languages for RDF has been since identified and deemed particularly important in the context of exchange of data between applications [Gayo *et al.* 2017, W3C 2013]. Having schema allows to validate RDF graphs and 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*. This class includes *shape constraint language* (SHACL) [Knublauch & Kontokostas 2017, Corman *et al.* 2018] and *shape expressions schemas* (ShEx) [Prud'hommeaux *et al.* 2018, Boneva *et al.* 2017, Staworko *et al.* 2015]. 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 types :TEmp and :TDept have the following ShEx definition

```
:TDept  { :name xsd:Literal; :address xsd:Literal}
:TEmp   { :name xsd:Literal; :email xsd:Literal?; :works @:TDept+ }
```

(where xsd:Literal is a new datatype that we introduce to XSD vocabulary). Essentially, every department IRI must have exactly one :name and :address property; and every employee IRI must have a single :name property, an optional :email property, and at least one :works property each leading to a department IRI (satisfying type :TDept).

We are considering the task of converting databases to RDF graphs. It is typically accomplished by a declarative formalism such as R2RML, DM, or YARRRML. Because of the importance of schema for RDF, we also require a target schema in any of the

declarative formalisms. As a consequence of this lastly consideration, the following concerns arise:

1. Conceptual complexity of designing mappings in particular when schemas are large can become easily an error-prone process. Can we offer static analysis tools for identification of design errors.

2. The existence of a schema and the fact that mappings are expressed in a declarative way makes possible existence of different solutions to a given input. Which solution is most suitable and if it exists, can it be constructed, what can be said about querying such a solution.

3. The mapping needs to be specified using a formal language which can be challenging to a non-expert user. Can we render this process more accessible while maintaining expressivity.

4. Many already existing solutions of exporting relational to RDF data only consider relational schema and mappings with no target schema. Can we propose a way to construct a schema and what is the good schema to the output graphs.

Formalization

We approach the above concerns by first formalizing the export of relational data to RDF as a data exchange problem. Data exchange has been mainly studied in the context of relational databases [Kolaitis 2005, Arenas *et al.* 2010, Barceló 2009]. Data exchange from relational databases to RDF has been studied recently [Sequeda *et al.* 2012, Boneva *et al.* 2015], but with little to no schema information for the output RDF graph. None of these previous works can be used to address precisely the setting we have at hand.

In this manuscript, we formalize the process of exporting a relational database to RDF with target schema as a *constructive data exchange setting* where we use source-to-target generating dependencies as mappings. These mappings use an abstraction of IRI templates which we call *IRI constructors* to map entities from the relational database to IRIs in the RDF.

R2RML mappings presented at the beginning of this section can be expressed with the following logical formalism

$$Dept(did, name) \Rightarrow Triple(dep2iri(did), \text{:name}, name) \wedge \mathsf{TDept}(dep2iri(did)),$$
$$Emp(eid, name, did) \wedge Email(eid, email) \Rightarrow$$
$$Triple(emp2iri(eid), \text{:name}, name) \wedge Triple(emp2iri(eid), \text{:email}, email) \wedge$$
$$Triple(emp2iri(eid), \text{:works}, dep2iri(did)) \wedge \mathsf{TEmp}(emp2iri(eid)),$$

where $dep2iri(did) = \texttt{"dept:}\{did\}\texttt{"}$ constructs an IRI for each department and the other IRI constructor $emp2iri(eid) = \texttt{"emp:}\{eid\}\texttt{"}$ generates an IRI for each employee.

This formalization trivially captures direct mapping (DM) [Arenas *et al.* 2012], large fragment of R2RML, and YARRRML since it is a succinct syntax for R2RML. Indeed, this formalization can express all four uses cases of R2RML [Auer *et al.* 2010] and it can cover 38 out of 54 test cases for R2RML implementations [Villazón & Hausenblas 2012]. The test cases that we do not cover are those that use pattern-based function to transform data values and we do not cover those that use SQL statements with aggregation functions. With this formalization, we address 8 out of the 11 core function requirements for R2RML [Auer *et al.* 2010]. A relatively straightforward extension of this formalization to node datatypes can cover 10 out of the 11 core function requirements. However, for relative simplicity, we do not study our formalization with datatypes.

Problems of interest

In this manuscript, we employ the above formalization and address the previously identified concerns by studying the following problems of interest.

Checking Consistency. A classical static analysis problem is testing consistency: checking whether for a given source instance of a relational schema there will be always a well-defined target instance. If the setting is inconsistent then there is possible error in the designing of mapping. Then, the static analysis allows to find errors and if it identi-

fies conditions that generate inconsistency then it is conceivable to extend to other kind of static analysis. This problem is complex because the source and target languages express constraints differently in an incompatible way. Additionally the mappings can add new constraints. In fact, we can view the problem of checking consistency as a variant of the problem of language containment i.e., does the language of the shapes schema contains the language of graphs produced from the source schema by applying the mappings.

Certain query answering and universal solutions. Certain answers allow to identify the important answers because they are present in every solution. As such, we are interested in finding a good solution that has the property of allowing to easily compute certain answers. Thus, we study the problem of certain query answering where certain answers typically are computed on a universal solution because it preserves information [Fagin *et al.* 2005a]. However, in the context of relational to RDF data exchange, a finite universal solution might not exist even if the setting is consistent and admits solutions. Thus, an interesting problem is to check if it is possible to construct a finite solution that has similar properties to a universal solution and to identify the family of queries for which it can be used to compute certain query answers. Also, we are interesting in a low complexity for the construction of this finite solution.

Visual mapping language. It has been argued [Shneiderman 1983] that a simple graphical interface facilitates the specification of mappings by the direct manipulation with drag-and-drop techniques and a visual language. Those kind of visual mapping languages are known to be easy for non-expert users as evidenced by Clio system [Fagin *et al.* 2009]. The challenge in this problem comes in identifying the trade-off between simplicity and expressivisity. The goal is to provide an interface based on a visual mapping language that is easy to use but does not hinder users with unnecessary details while allows to cover the most possible number of use cases or it is expressible as possible.

Schema elicitation. Shapes schema elicitation aims at constructing a target schema from a set of mappings and a relational schema. An important issue in this construction is to define the properties of the desired target schema because there can be trivial solutions that produce a schema. For instance, solutions that produce universal schemas that accept any graph and not only graphs produced by the mapping. We want an algorithm

that makes an honest effort in capturing the structure of the graphs produced.

Contributions

In this section, we describe the contributions presented to solve the problems above.

Consistency. We have solved the problem of consistency by identifying a set of necessary and sufficient conditions that guarantees the consistency. This characterization aims at constructing an inconsistent instance. One such condition is *value consistency* that guarantees that no two different values are equated to satisfy the shapes schema. For instance, in the previous example, an employee that has two different names is an example of value inconsistency. On the other hand, a mapping rule may declare the type of nodes it constructs as IRI or literal. A malformed set of rules may inconspicuously produce a graph with a node whose declared type is both IRI node and literal node. The condition of *node kind consistency* guarantees the no co-occurrence of literal and non literal types for a node. We have developed an static analysis tool that allows to decided if a setting is consistent. The complexity of checking those conditions is coNP-complete. Initial findings have been published in AMW [Boneva *et al.* 2018] and further developed in ADBIS [Boneva *et al.* 2020].

Certain query answering. We have proposed a novel notion of universal simulation solution, which is an adaptation of universal solution by replacing homomomorphisms with a weaker notion of simulation. A universal simulation solution always exists and it allows to compute certain answers to classes of queries that are robust under simulation. Intuitively, a robust query class is when a query is evaluated on a node, the query will behave in a same way under any node that is simulated. We can construct a minimal-size universal simulation solution to a given instance of the relational schema w.r.t. a constructive setting is exponential in the size of the shapes schema. If we consider the class of nested regular expressions, which are used as a navigational query language by SPARQL [Pérez *et al.* 2010] to query RDF, then the forward fragment is robust under simulation. The data complexity of computing certain answers for nested regular expressions and any constructive data exchange setting is PTIME. We have published this result in ADBIS [Boneva *et al.* 2020].

Visual mapping language. We have proposed a visual mapping language (VML) that covers a large fragment of constructive relational to RDF data exchange setting. Also, we have developed ShERML, a tool that guides a user through interactive functionalities in the process of exporting relational data to RDF. In addition, we have proposed R2VML, a declarative mapping language that captures VML mappings and presents a user-friendly and succinct syntax for defining mappings. We have published the result related to the visual mapping language in ISWC [Boneva *et al.* 2019].

Shapes schema elicitation. We have investigated the problem of shapes schema elicitation and we have identified two desirable properties of the target schema, which are *soundness* i.e., every produced RDF graph is accepted by the target schema; and *completeness* i.e., every RDF graph accepted by the target schema can be produced. We have proposed an algorithm called M3, based on a method of minimal and maximal models, that produces a sound schema for any relational schema and set of mappings. We have shown that for relational schemas and mappings that are the result of the straightforward translation of a class of ER diagrams, the schema produced by M3 is also complete. Finally, we have presented two inherent limitations of the task at hand: a sound and complete schema might be of exponential size or worst even it might not exist. We are currently preparing for publication.

Organization

In this section, we describe briefly the content of each chapter of the manuscript.

Preliminaries. Chapter 1 provides the basic definitions and notations used throughout this manuscript. First, we recall the notions of signatures, models, semantics and classes of formulas. Second, *relational databases* and *dependencies* are introduced with the definition and formalization of them. Third, we describe the concepts that are related to *relational data exchange setting*, which are *chase*, *universal solution*, *certain query answering* and *consistency*. Fourth, RDF is presented and formalized. Finally, we present the shapes constraints language and show how these constraints can be expressed with dependencies.

Relational to RDF data exchange. Chapter 2 illustrates a relational to RDF data ex-

change setting. We present its formalization and define what is a solution to our setting. Also, we present R2RML and show which part of it is captured by our setting, and how this is captured. We end with an introductory presentation of the problems of interest.

Consistency. Chapter 3 provides the study of consistency problem. First, we identify the sources of inconsistency by an example. Then, we present two conditions that will guarantee the absence of the source of inconsistency. Each condition is shown to be necessary for deciding consistency and an algorithm is presented for each condition to identify if the setting satisfies it. At the end of the chapter, we show that consistency is decidable.

Certain query answering. Chapter 4 provides the study of certain query answering problem. We start with an adaptation of two existing approaches to the problem of certain query answering. Then, we present our approach that investigates forward nested regular expressions and introduces the notion of a universal simulation solution that allows to compute certain answers efficiently. We present the construction of a size-minimal universal simulation solution and show that its size is polynomial in the size of the source instance, but might be exponential in the size of the constraints.

Visual mapping language. Chapter 5 provides a tool that defines visual mappings. The notions of human-computer interaction used to develop the tool are described at the beginning. Then, a text-based mapping language is defined and on top of this language is defined a visual mapping language. Based on these languages, we describe the tool developed for facilitating the definition of mappings in a relational to RDF data exchange context. We do an evaluation of the tool and present its results.

Shapes schema elicitation. Chapter 6 defines the shapes schema elicitation problem where the notions of soundness and completeness of a desirable target schema are defined. An elicitation algorithm is proposed based on minimal and maximal models described in the same section. Then, we show the soundness of the algorithm. We identify a class where the proposed algorithm is complete. Finally, we present inherent limitations of the task of producing sound and complete schema.

Chapter 1

Preliminaries

In this chapter, we recall the basic concepts and introduce the notation that will be used throughout this manuscript. We start in Section 1.1 by recapping fundamentals of logic. In Section 1.2, we recall basic notions of relational databases. Next, we describe a general notion of relational data exchange in Section 1.3. Then we describe RDF graphs and their formalization in Section 1.4. Finally, in Section 1.5, we discuss the notion of schemas for RDF graphs and introduce an abstraction formalism that captures a common fragment of two well-known schemas ShEx and SHACL.

1.1 Logic

We fix an enumerable set Dom of *constant values*. Because we deal with relational databases and RDF graphs in this manuscript, we need to distinguish for RDF, nodes and literals where nodes and literals can have null values; and we need to distinguish for relational databases, constants and null values. Thus, we specify the domain to be partitioned into three infinite subsets $\mathsf{Dom} = \mathsf{Iri} \cup \mathsf{Lit} \cup \mathsf{Blank}$, of *IRIs*, *literals*, and *blank* node identifiers respectively. Because relational databases deal with null values that are not blank nodes, we assume an infinite set of *null literals* $\mathsf{NullLit} \subseteq \mathsf{Lit}$ for relational databases; and the set of all *null values* $\mathsf{Null} = \mathsf{NullLit} \cup \mathsf{Blank}$. We refer to the remaining elements as *constants* $\mathsf{Const} = \mathsf{Dom} \setminus \mathsf{Null}$, and additionally, because relational databases use constants that are neither IRIs nor null literals, we identify non-null literals $\mathsf{ConstLit} = \mathsf{Lit} \setminus \mathsf{NullLit} = \mathsf{Const} \cap \mathsf{Lit}$.

The *relational vocabulary* is an enumerable set of symbols $\mathscr{V} = \mathscr{V}_a \cup \mathscr{V}_R \cup \mathscr{V}_f$

partitioned into three pair-wise disjoint subsets: the set of *first-order names* $\mathscr{V}_a$, the set of *relation names* $\mathscr{V}_R$, and the set of *function names* $\mathscr{V}_f$. In our formulas, we are going to use names of the constant values of Dom. So, we specify the set of first-order names, $\mathscr{V}_a = \mathscr{V}_{\mathscr{V}} \cup \mathsf{Dom}$ as a set of variable names $\mathscr{V}_{\mathscr{V}}$ and a set of names of the domain corresponding to IRIs, literals and blank nodes.

For a function name or relational name, the number of inputs is called the arity of the function or relational name. We assume that each relation and function name come with a fixed arity. In the sequel, by $\mathscr{V}_R^{(m)}$ we denote the set of relation names of arity m, and analogously, by $\mathscr{V}_f^{(n)}$ we denote the set of function names of arity n. Every element of the vocabulary has its *domain*, the set of compatible values:

$$dom(v) = \mathsf{Dom}, \quad dom(R) = 2^{\mathsf{Dom}^m}, \quad dom(f) = \mathsf{Dom}^{\mathsf{Dom}^n}, \quad dom(c) = \{c\}$$

for any $v \in \mathscr{V}_{\mathscr{V}}$, $c \in \mathsf{Dom}$, $R \in \mathscr{V}_R$ with arity m, and for any $f \in \mathscr{V}_f$ with arity n. We point out that the domain of a constant symbol is the same constant value.

Syntax. A *term* is defined by the following recursive syntax:

$$t ::= a \mid f(t, \ldots, t)$$

where $a \in \mathscr{V}_a$, and $f \in \mathscr{V}_f$. A *formula* is defined with the following syntax:

$$\varphi ::= R(t, \ldots, t) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists v.\, \varphi \mid t = t$$

where $R \in \mathscr{V}_R$, t is a term, and v ranges over $\mathscr{V} \setminus \mathscr{V}_f$. We use the following syntactic-sugar:

$$\forall v.\, \varphi ::= \neg\exists v.\, \neg\varphi, \qquad \varphi \vee \psi ::= \neg(\neg\varphi \wedge \neg\psi), \qquad \varphi \Rightarrow \psi ::= \neg\varphi \vee \psi.$$

Signatures and models. A *relational signature* $\mathscr{R} \subseteq \mathscr{V}$ is a finite set of symbols. A *relational structure* of $\mathscr{R}$, or a *model* of $\mathscr{R}$, is a function M that assigns to every symbol $v \in \mathscr{R}$ a compatible value $v^M \in dom(v)$. In essence, relational signature identifies a set of names whose interpretation is provided by a model. The *size of a signature* $\mathscr{R}$, denoted by $|\mathscr{R}|$, is the number of symbols in the signature.

The *union* of two models M_1 of $\mathscr{R}_1$ and M_2 of $\mathscr{R}_2$, whose signatures are disjoint $\mathscr{R}_1 \cap \mathscr{R}_2 = \emptyset$, is a model $M_1 \cup M_2$ of $\mathscr{R}_1 \cup \mathscr{R}_2$ defined as follows (for any $v \in \mathscr{R}_1 \cup \mathscr{R}_2$):

$$v^{M_1 \cup M_2} = \begin{cases} v^{M_1} & \text{if } v \in \mathscr{R}_1, \\ v^{M_2} & \text{if } v \in \mathscr{R}_2. \end{cases}$$

Given two models M_1 of $\mathscr{R}_1$ and M_2 of $\mathscr{R}_2$, M_1 is *compatible* with M_2 if for all $v \in \mathscr{R}_1 \cap \mathscr{R}_2$, $v^{M_1} = v^{M_2}$. The *merge* of two compatible models M_1 of $\mathscr{R}_1$ and M_2 of $\mathscr{R}_2$ is a model $M_1 \uplus M_2$ of $\mathscr{R}_1 \cup \mathscr{R}_2$ defined as follows (for any $v \in \mathscr{R}_1 \cup \mathscr{R}_2$):

$$v^{M_1 \uplus M_2} = \begin{cases} v^{M_1} & \text{if } v \in \mathscr{R}_1, \\ v^{M_2} & \text{if } v \in \mathscr{R}_2 \setminus \mathscr{R}_1. \end{cases}$$

A term over a signature $\mathscr{R}$ is defined with

$$t ::= a \mid f(t, \ldots, t)$$

where $a \in \mathscr{V}_a \cap \mathscr{R}$, and $f \in \mathscr{V}_f \cap \mathscr{R}$. A formula over a signature $\mathscr{R}$ is defined with:

$$\varphi ::= R(t, \ldots, t) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists v.\ \varphi \mid t = t$$

where $R \in \mathscr{V}_R \cap \mathscr{R}$, t is a term over $\mathscr{R}$, and $v \in \mathscr{V}_\mathscr{V} \cup (\mathscr{V}_a \cap \mathscr{R})$. Implicitly, all signatures will allow to use equality symbol $=$.

Classes of formulas. We point out that any symbol in $\mathscr{V}$ can serve the purpose of a variable, in particular the symbol present in a relational signature $\mathscr{R}$ can also be bound with a quantifier (overriding whatever interpretation the structure assigns to this symbol). We identify the set of symbols used in a term as follows:

$$\begin{aligned} &Vocab(a) = \{a\} \\ &Vocab(f(t_1, \ldots, t_n)) = \{f\} \cup \textstyle\bigcup_{i=1}^{n} Vocab(t_i), \end{aligned}$$

and the set of symbols used in a formula,

$$Vocab(R(t_1,\ldots,t_m)) = \{R\} \cup \bigcup_{i=1}^{m} Vocab(t_i), \qquad Vocab(\neg\varphi) = Vocab(\varphi),$$
$$Vocab(\varphi_1 \wedge \varphi_2) = Vocab(\varphi_1) \cup Vocab(\varphi_2), \qquad Vocab(\exists v.\ \varphi) = Vocab(\varphi),$$
$$Vocab(t_1 = t_2) = Vocab(t_1) \cup Vocab(t_2).$$

Similarly, we can identify the set of unbound symbols used in a formula

$$Unbd(R(t_1,\ldots,t_m)) = \{R\} \cup \bigcup_{i=1}^{m} Vocab(t_i), \quad Unbd(\neg\varphi) = Unbd(\varphi),$$
$$Unbd(\varphi_1 \wedge \varphi_2) = Unbd(\varphi_1) \cup Unbd(\varphi_2), \qquad Unbd(\exists v.\ \varphi) = Unbd(\varphi) \setminus \{v\},$$
$$Unbd(t_1 = t_2) = Vocab(t_1) \cup Vocab(t_2)$$

Now, let $\mathscr{R}$ be a relational signature and φ a formula over $\mathscr{R}$. The set of *variables* used in φ is $vars_{\mathscr{R}}(\varphi) = Vocab(\varphi) \setminus \mathscr{R}$. The set of *free variables* of a formula φ is $fvars_{\mathscr{R}}(\varphi) = Unbd(\varphi) \setminus \mathscr{R}$. The formula φ is *closed* iff it has no free variable i.e., $fvars_{\mathscr{R}}(\varphi) = \emptyset$. An *atomic* formula has the form $R(t,\ldots,t)$, where $R \in \mathscr{V}_R$. A formula is *ground* if it uses no variables whatsoever. A *relational atom* does not use any function symbols. A *fact* is a ground relational atom. A *first-order* (FO) formula over $\mathscr{R}$ uses only first-order variables i.e., $vars_{\mathscr{R}}(\varphi) \subseteq \mathscr{V}_a$. A *second-order* (SO) formula over $\mathscr{R}$ uses second-order variables in addition to first-order variables i.e., $vars_{\mathscr{R}}(\varphi) \subseteq \mathscr{V}_a \cup \mathscr{V}_R \cup \mathscr{V}_f$. A *monadic SO logic* (MSO) formula over $\mathscr{R}$ uses no second-order functional variable and no second-order relational variable of arity higher than 1 i.e., $vars_{\mathscr{R}}(\varphi) \subseteq \mathscr{V}_R^{(1)} \cup \mathscr{V}_a$. An *existential second-order* ($\exists$SO) formula over $\mathscr{R}$ has the form $\varphi = \exists X_1,\ldots,X_n.\ \varphi$, where $X_1,\ldots,X_n \in \mathscr{V}_R \cup \mathscr{V}_f$, and φ does not have a quantifier with a second-order variable in $\mathscr{V}_R \cup \mathscr{V}_f$. If furthermore, all $X_1,\ldots,X_n \in \mathscr{V}_R^{(1)}$, then φ is an *existential monadic SO* ($\exists$MSO) formula.

Semantics. We define the *entailment relation* between a relational structure M of $\mathscr{R}$ and a formula φ over $\mathscr{R}$. First, we define an *expression* E over $\mathscr{R}$ as any term, variable name, function name or relation name. We identify the free variables $V \subseteq fvars_{\mathscr{R}}(E)$ of an expression. A *valuation* of V is a function θ that takes a symbol $v \in V$ and yields a value in $dom(v)$. By $\emptyset$ we denote the *empty valuation* of the empty set of symbols. Given a valuation θ of V, a symbol $v \in \mathscr{V} \setminus \mathscr{R}$, and a value $w \in dom(v)$, $\theta[v/w]$ is

valuation θ' of $V \cup \{v\}$ such that $\theta'(v) = w$ and $\theta'(z) = \theta(z)$ for any $z \in V \setminus \{v\}$. Given a relational signature $\mathscr{R}$, a model M of $\mathscr{R}$, an expression E over $\mathscr{R}$ and a valuation θ of V such that $fvars_{\mathscr{R}}(E) \subseteq dom(\theta)$ where dom returns the domain of θ, the *interpretation* of E w.r.t. θ and M is defined as follows:

$$v^{(M,\theta)} = \begin{cases} v^M & \text{if } v \in \mathsf{Dom} \\ \theta(v) & \text{Otherwise.} \end{cases} \qquad \text{for any } v \in V,$$

$$R^{(M,\theta)} = \begin{cases} R^M & \text{if } R \notin dom(\theta) \\ \theta(R) & \text{Otherwise.} \end{cases} \qquad \text{for any } R \in V,$$

$$f^{(M,\theta)} = \begin{cases} f^M & \text{if } f \notin dom(\theta) \\ \theta(f) & \text{Otherwise.} \end{cases} \qquad \text{for any } f \in V,$$

$$f(t_1, \ldots, t_n)^{(M,\theta)} = f^{(M,\theta)}(t_1^{(M,\theta)}, \ldots, t_n^{(M,\theta)})$$

$$t^{(M,\theta)} = \begin{cases} v^{(M,\theta)} & \text{if } t = v, \\ f(t_1, \ldots, t_n)^{(M,\theta)} & \text{Otherwise.} \end{cases} \qquad \text{for any term } t.$$

Below, we define the *entailment relation* $(M, \theta) \models \varphi$ of a formula φ over $\mathscr{R}$ w.r.t. a model M of $\mathscr{R}$ and a valuation such that $dom(\theta) \subseteq fvars_{\mathscr{R}}(\varphi)$ is defined as follows:

$$\begin{array}{lll} (M,\theta) \models R(t_1, \ldots, t_n) & \text{iff} & (t_1^{(M,\theta)}, \ldots, t_n^{(M,\theta)}) \in R^{(M,\theta)}, \\ (M,\theta) \models \neg\varphi & \text{iff} & \text{it is not true that } (M,\theta) \models \varphi, \\ (M,\theta) \models (\varphi_1 \wedge \varphi_2) & \text{iff} & (M,\theta) \models \varphi_1 \text{ and } (M,\theta) \models \varphi_2, \\ (M,\theta) \models \exists v.\, \varphi & \text{iff} & (M, \theta[v/w]) \models \varphi \text{ for some } w \in dom(v). \\ (M,\theta) \models t_1 = t_2 & \text{iff} & (M,\theta) \models t_1^{(M,\theta)} = t_2^{(M,\theta)}. \end{array}$$

Finally, we say that φ is satisfied in M, in symbols $M \models \varphi$ iff $(M, \emptyset) \models \varphi$.

1.2 Relational databases

We recall the basic notions of relational databases [Garcia-Molina *et al.* 2009]. A relational database is a *collection of tables* where each table has a name and each column

of a table also has a name called *attribute*. From the modeling point of view, a table is seen as a relation. Each table gives some structural information on the relation, and the whole structural information of all relations is called the *schema*. Each row of the table is called a *tuple* and stores the attribute values. The set of tuples is the content of a table and the content of the whole database is called an *instance*. The schema also includes a set of constraints, which defines consistency of the instance. Here, we focus only on *functional dependencies* and *inclusion dependencies*. We illustrate the above notions with the following example.

Example 1.2.1. Consider the database in Figure 1.1 of an academic institute that organizes conferences. Figure 1.1 shows an instance together with a schema including *primary keys* which are the attributes that are underlined and *foreign keys* which are represented with arrows. This database stores information structured in a set of *tables* about researchers that have attended conferences, presented talks and for each talk stores the collaborators. This information is structured in the following tables:

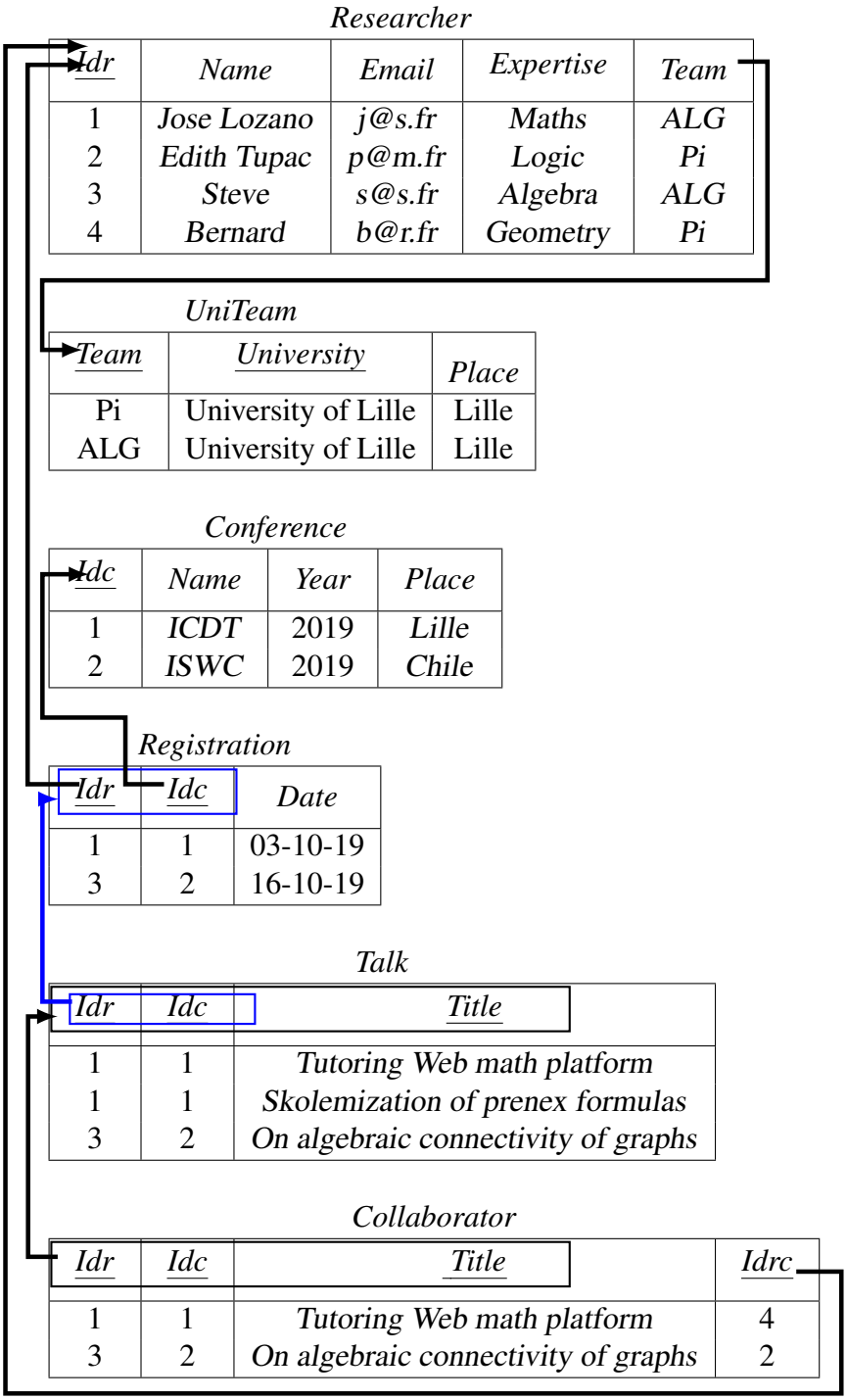
- Table *Researcher* stores researchers with their name, email, expertise and team.
- Table *UniTeam* stores teams associated to a university and located in a place.
- Table *Conference* stores conferences with their name, year and place.
- Table *Registration* stores the date when a researcher is registered in a conference.
- Table *Talk* stores the title of a researcher talk done in a conference;
- Table *Collaborator* stores *tuples* that indicate a researcher that has collaborated in a paper presented in a talk.

The schema of the conference database is composed of the relations that are the tables presented above and defines the following set of *constraints*:

(R1) every researcher and every conference have a unique identifier;

(R2) every team of a university is identified by a unique name;

(R3) every collaborator is identified by a unique research ID, conference ID, title, and other research ID;



Researcher

Idr	*Name*	*Email*	*Expertise*	*Team*
1	*Jose Lozano*	*j@s.fr*	*Maths*	*ALG*
2	*Edith Tupac*	*p@m.fr*	*Logic*	*Pi*
3	*Steve*	*s@s.fr*	*Algebra*	*ALG*
4	*Bernard*	*b@r.fr*	*Geometry*	*Pi*

UniTeam

Team	*University*	*Place*
Pi	University of Lille	Lille
ALG	University of Lille	Lille

Conference

Idc	*Name*	*Year*	*Place*
1	*ICDT*	2019	*Lille*
2	*ISWC*	2019	*Chile*

Registration

Idr	*Idc*	*Date*
1	1	03-10-19
3	2	16-10-19

Talk

Idr	*Idc*	*Title*
1	1	*Tutoring Web math platform*
1	1	*Skolemization of prenex formulas*
3	2	*On algebraic connectivity of graphs*

Collaborator

Idr	*Idc*	*Title*	*Idrc*
1	1	*Tutoring Web math platform*	4
3	2	*On algebraic connectivity of graphs*	2

Figure 1.1: Database instance of conference schema.

(R4) every registration and every talk are identified by a unique pair of research ID and conference ID;

(R5) every registration has a reference to the relation *Researcher* and a reference to the relation *Conference*;

(R6) every talk has a reference to the relation *Registration*;

(R7) every collaborator has a reference to the relation *Talk* and a reference to the relation *Researcher*; and

(R8) every researcher has a reference to the relation *UniTeam*.

We can create the schema of the conference database in RDBMS using SQL syntax as shown in Figure 1.2. The primary keys in RDBMS are rules from R1 to R4, and the foreign keys are rules from R5 to R8. □

```
CREATE TABLE UniTeam (
  Team varchar,
  University varchar,
  Place varchar,
  PRIMARY KEY (Team,University)
);
CREATE TABLE Conference (
  Idc int,
  Name varchar,
  Year date,
  Place varchar,
  PRIMARY KEY(Idc)
);
CREATE TABLE Registration (
  Idr int,
  Idc int,
  Date date,
  PRIMARY KEY (Idr,Idc),
  FOREIGN KEY (Idr)
    REFERENCES Researcher(Id),
  FOREIGN KEY (Idc)
    REFERENCES Conference(Id)
);
CREATE TABLE Talk (
  Idr int,
  Idc int,
  Title varchar,
  PRIMARY KEY (Idp,Idc,Title),
  FOREIGN KEY (Idr,Idc)
    REFERENCES Registration(Idr,Idc)
);
```

```
CREATE TABLE Researcher (
  Idr int,
  Name varchar,
  Email varchar,
  Expertise varchar,
  Team varchar,
  PRIMARY KEY(Idr),
  FOREIGN KEY (Team)
     REFERENCES UniTeam(Team)
);
CREATE TABLE Collaborator (
  Idr int,
  Idc int,
  Title varchar,
  Idrc int,
  PRIMARY KEY (Idr,Idc,Title,Idrc),
  FOREIGN KEY (Idr,Idc,Title)
     REFERENCES Talk(Idr,Idc,Title),
  FOREIGN KEY (Idrc)
     REFERENCES Researcher(Id)
);
```

Figure 1.2: Relational schema creation with SQL syntax.

1.2.1 Relational schema and dependencies

We assume an infinite set of *attribute names* $\mathscr{A} \subseteq \mathscr{V}_a$. A *relational schema* is a tuple $\mathbf{R} = (\mathscr{R}, attrs, \Sigma_{\text{fd}}, \Sigma_{\text{ind}})$ where

- $\mathscr{R}$ is a finite set of relation names, each relation name with a given arity;
- $attrs$ is a function $attrs{:}\mathscr{R} \to \mathscr{P}(\mathscr{A})$ that assigns to each relation name $R \in \mathscr{R}$ a finite set of attribute names whose cardinality is equal to the arity of R;
- Σ_{fd} is a set of *functional dependencies* (fds) of the form $R : X \to Y$, where $R \in \mathscr{R}$ is a relation name, and $X, Y \subseteq attrs(R)$, and
- Σ_{ind} is a set of *inclusion dependencies* of the form $R[X] \subseteq P[Y]$, where $R, P \in \mathscr{R}$ are relation names, $X \subseteq attrs(R)$ and $Y \subseteq attrs(P)$ and $|X| = |Y|$.

Also, we assume that the set of attribute names of every relation have a fixed order.

Intuitively, a fd $R : X \to Y$ means that values of attribute names in X determine the values of attribute names in Y in every tuple; and an inclusion dependency between two relations R and P, $R[X] \subseteq P[Y]$, means that every value of an attribute in X is the same value of an attribute in Y. We point out that *key dependencies* (kds) are a special case of functional dependencies where $Y = attrs(R)$. Also, foreign key constraints are a special case of inclusion dependencies where Y is a primary key in P.

Given a finite set $A \subseteq \mathscr{A}$ of attribute names, a *tuple* over A is a function that maps from an attribute name to a value, i.e., $t : A \to \mathsf{Dom}$. We view a tuple as a record, consequently we write $t.a$ for $t(a)$. Let $X \subseteq A$ be a set of attributes, by $t.X$ we denote a tuple over X that consists of precisely the value of t on attributes in X, i.e., $t : X \to \mathsf{Dom}$ and $t.X(a) = t(a)$ for $a \in X$. An *instance* I of a relational schema $\mathbf{R}$ is a function I that maps every relation name $R \in \mathscr{R}$ to a finite set $I(R)$ of tuples over attributes of R. The *size of an instance* I of $\mathscr{R}$, denoted by $|I|$ is the sum of number of tuples by relation mapped by I.

An instance I of a relational schema $\mathbf{R}$ *satisfies* a fd $R : X \to Y$, denoted by $I \models R : X \to Y$, if for any pair of tuples in $I(R)$ that agree on X also agree on Y. Formally, $I \models R : X \to Y$ iff $\forall t_1, t_2 \in I(R).\ t_1.X = t_2.X \Rightarrow t_1.Y = t_2.Y$.

An instance I of a relational schema $\mathbf{R}$ *satisfies* an inclusion dependency $R[X] \subseteq P[Y]$, denoted by $I \models R[X] \subseteq P[Y]$, if for any tuple $t \in I(R)$, there is a tuple $t' \in I(P)$ such that the tuple t over X agrees with t' over Y. Formally, $I \models R[X] \subseteq P[Y]$ iff $\forall t \in I(R).\ \exists t' \in I(P).\ ran(t) \subseteq ran(t') \wedge t.X = t'.Y$ where ran returns the range of t. Finally, we say that an instance I is *consistent* if I satisfies the set of dependencies

$\Sigma_{\text{fd}} \cup \Sigma_{\text{ind}}$. The *active domain* $adom(I)$ of the instance I is the set of values from Dom used in I. For ease of use, when writing a functional dependency or inclusion dependency, we write a set of attributes $A = \{a_1, \ldots, a_n\}$ as $A = a_1 \ldots a_n$. We denote the set of instances of a relational schema $\mathbf{R}$ by $Inst(\mathbf{R})$.

Example 1.2.2 (cont. Example 1.2.1)**.** Take the conference database, the relational schema $\mathbf{R}_0 = (\mathscr{R}_0, attrs, \Sigma_{\text{fd}}, \Sigma_{\text{ind}})$ has the following relation names

$$\mathscr{R}_0 = \{\textit{Researcher}, \textit{UniTeam}, \textit{Conference}, \textit{Registration}, \textit{Talk}, \textit{Collaborator}\},$$

and we express the set of constraints with the following dependencies $\Sigma_{\text{fd}} \cup \Sigma_{\text{ind}}$:

$$\textit{Researcher} : Idr \rightarrow Idr\ Name\ Email\ Expertise\ Team \quad \text{(R1)}$$

$$\textit{Conference} : Idc \rightarrow Idc\ Name\ Year\ Place \quad \text{(R1)}$$

$$\textit{UniTeam} : Team\ University \rightarrow Team\ University\ Place \quad \text{(R2)}$$

$$\textit{Collaborator} : Idr\ Idc\ Title\ Idrc \rightarrow Idr\ Idc\ Title\ Idrc \quad \text{(R3)}$$

$$\textit{Registration} : Idr\ Idc \rightarrow Idr\ Idc\ Date \quad \text{(R4)}$$

$$\textit{Talk} : Idr\ Idc\ Title \rightarrow Idr\ Idc\ Title \quad \text{(R4)}$$

$$\textit{Registration}[Idc] \subseteq \textit{Conference}[Idc] \quad \text{(R5)}$$

$$\textit{Registration}[Idr] \subseteq \textit{Researcher}[Idr] \quad \text{(R5)}$$

$$\textit{Talk}[Idr\ Idc] \subseteq \textit{Registration}[Idr\ Idc] \quad \text{(R6)}$$

$$\textit{Collaborator}[Idr\ Idc\ Title] \subseteq \textit{Talk}[Idr\ Idc\ Title] \quad \text{(R7)}$$

$$\textit{Collaborator}[Idrc] \subseteq \textit{Researcher}[Id] \quad \text{(R7)}$$

$$\textit{Researcher}[Team] \subseteq \textit{UniTeam}[Team] \quad \text{(R8)}$$

We recall that the set of functional dependencies in this example are primary keys and the inclusion dependencies are foreign key constraints. □

1.2.2 Logic formalization

The *relational signature* of the relational schema $\mathbf{R} = (\mathscr{R}, attrs, \Sigma_{\text{fd}}, \Sigma_{\text{ind}})$ is the set of relation names $\mathscr{R}$. For a given relational schema $\mathbf{R}$ and a database instance I of $\mathbf{R}$,

by M_I we define the relational structure over $\mathscr{R}$ corresponding to I with $R^{M_I} = I(R)$ for all $R \in \mathscr{R}$. In the sequel, we treat database instances interchangeably with their relational structures, and unless we state otherwise, we work only with instances that use literal constants from ConstLit.

In general, given a relational schema $\mathbf{R} = (\mathscr{R}, attrs, \Sigma_{\text{fd}}, \Sigma_{\text{ind}})$, a *database constraint* either functional or inclusion dependency is modeled by a *dependency*, which is any closed formula σ over $\mathscr{R}$ of the form

$$\forall \mathbf{x}, \mathbf{y}.\ \varphi(\mathbf{x}, \mathbf{y}) \Rightarrow \exists \mathbf{z}.\ \psi(\mathbf{y}, \mathbf{z})$$

where

- $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are sequences of variables,
- $\varphi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathscr{R}$ and is called the body of σ, denoted as $body(\sigma)$; and
- $\psi(\mathbf{y}, \mathbf{z})$ is called the head of σ, denoted as $head(\sigma)$, and is either a conjunction of equality terms or a conjunction of atomic formulas over $\mathscr{R}$.

We distinguish two types of dependencies by the form of $head(\sigma)$:

- *equality generating dependency (egd)* if the dependency is a conjunction of equality terms; and
- *tuple generating dependency (tgd)* if the dependency is a conjunction of atomic formulas.

In the sequel, we write $\mathbf{x}$ to denote a sequence of variables and for writing dependencies, we often drop the universal quantifier and write simply $\varphi(\mathbf{x}, \mathbf{y}) \Rightarrow \exists \mathbf{z}.\ \psi(\mathbf{y}, \mathbf{z})$; and we assume that implicitly all free variables are universally quantified.

Any inclusion dependency can be expressed with tgds and any fd is in fact an egd. For instance, the following example illustrates how fds and inclusion dependencies can be expressed with tgds and egds.

Example 1.2.3 (cont. Example 1.2.2)**.** We express the functional dependencies with the

following formulas over $\mathscr{R}_0$:

$$\textit{Researcher}(x_1, y_2, y_3, y_4, y_5) \wedge \textit{Researcher}(x_1, z_2, z_3, z_4, z_5) \Rightarrow \bigwedge_{i=2}^{5} y_i = z_i,$$
$$\textit{Registration}(x_1, x_2, x_3) \wedge \textit{Registration}(x_1, x_2, y_3) \Rightarrow x_3 = y_3,$$
$$\textit{Conference}(x_1, y_2, y_3, y_4) \wedge \textit{Conference}(x_1, z_2, z_3, z_4) \Rightarrow y_2 = z_2 \wedge y_3 = z_3 \wedge y_4 = z_4,$$
$$\textit{Talk}(x_1, x_2, x_3) \wedge \textit{Talk}(x_1, x_2, x_3) \Rightarrow x_1 = x_1 \wedge x_2 = x_2 \wedge x_3 = x_3.$$

We omit the key dependencies for *UniTeam* and *Collaborator* since there are similar to *Talk* because they do not imply other attributes. Now, we express the inclusion dependencies with the following formulas over $\mathscr{R}_0$:

$$\textit{Registration}(x_1, x_2, x_3) \Rightarrow \exists y.\ \textit{Conference}(x_1, y),$$
$$\textit{Registration}(x_1, x_2, x_3) \Rightarrow \exists y_1, y_2, y_3, y_4.\ \textit{Researcher}(x_1, y_1, y_2, y_3, y_4),$$
$$\textit{Talk}(x_1, x_2, x_3) \Rightarrow \exists y.\ \textit{Registration}(x_1, x_2, y),$$
$$\textit{Collaborator}(x_1, x_2, x_3, x_4) \Rightarrow \exists y.\ \textit{Talk}(x_1, x_2, x_3),$$
$$\textit{Researcher}(x_1, x_2, x_3, x_4, x_5) \Rightarrow \exists y_1, y_2.\ \textit{UniTeam}(x_5, y_1, y_2). \qquad \square$$

Finally, an instance I of $\mathbf{R}$ satisfies a dependency σ if $I \models \sigma$.

1.2.3 Database queries

A database query extracts data from an instance of a relational schema. Such extracted data is called an *answer*. Typically, a query is modeled by a first-order formula. Here we also treat Boolean queries where the answers are $true$ or $false$. We illustrate an answer to a query in the next example.

Example 1.2.4 (cont. Example 1.2.2)**.** Recall the conference database in Example 1.2.2. Consider the query φ_1 which asks the talks of *Steve* on the *ISWC* conference in the year 2019:

$$\varphi_1(y) = \exists x_1, x_2, x_3, x_4, x_5, x_6.\ \textit{Researcher}(x_1, \textit{Steve}, x_2, x_3, x_4) \wedge$$
$$\textit{Conference}(x_5, \textit{ISWC}, 2019, x_6) \wedge \textit{Talk}(x_1, x_5, y). \quad (1.1)$$

The set of answers to φ_1 is $\{(\textit{Skolemization of prenex formulas})\}$.

Now, consider a Boolean query φ_2 which asks if *Steve* has presented a talk on a *ICDT* conference in the year 2019:

$$\varphi_2() = \exists x_1, x_2, x_3, x_4, x_5, x_6, x_7.\ \textit{Researcher}(x_1, \textit{Steve}, x_2, x_3, x_4) \wedge \textit{Conference}(x_5, \textit{ICDT}, 2019, x_6) \wedge \textit{Talk}(x_1, x_5, x_7). \quad (1.2)$$

The answer to this query is false. □

We define queries using formulas. Let $\mathbf{R} = (\mathscr{R}, \mathit{attrs}, \Sigma_{\text{fd}}, \Sigma_{\text{ind}})$ be a relational schema. Let I be a database instance of $\mathbf{R}$ and let φ be a first-order formula. A valuation val of $\mathit{fvars}_{\mathscr{R}}(\varphi)$ is an *answer* to a query φ in I if and only if $(I, \mathit{val}) \models \varphi$. The set of answers to a query φ in I is $QA(\varphi, I) = \{\mathit{val} \mid (I, \mathit{val}) \models \varphi\}$. If a query is a closed formula, the answers are Boolean: true if $QA(\varphi, I) = \{\emptyset\}$, otherwise is false. This kind of query is called *Boolean query*.

A *conjunctive query* is an existentially quantified conjunction of atomic formulas over a relational signature $\mathscr{R}$ and *Boolean conjunctive query* (BCQ) is conjunction of closed atomic formulas. The *size* of a conjunctive query Q is denoted by $|Q|$ and it represents the number of atoms in Q.

A conjunctive query φ is *acyclic* if it has a *join-tree*, otherwise is *cyclic*. A join-tree is a tree T such that nodes are the relational atoms of the conjunctive query φ and for every variable x of φ the set of relational atoms with x forms a subtree of T.

Example 1.2.5. Consider a query $\varphi(x, y) = \exists z, w.\ E(x, z) \wedge E(z, w) \wedge E(y, z)$. This query is acyclic because it has a join tree as seen in Figure 1.3. □

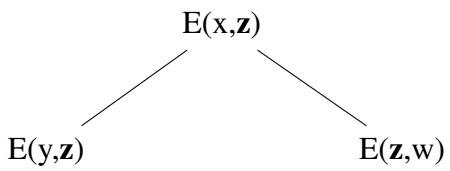


Figure 1.3: Join tree of a conjunctive query.

1.3 Relational data exchange

We recall the classical data management problem called *data exchange*. Relational data exchange is the transformation from a *source relational database* to a *target relational database*. Such a transformation is defined by a set of *mappings* from the source relational schema to the target schema. These elements constitute the *data exchange setting*. The result of a transformation of a source instance is an instance of the target schema, and if it satisfies the constraints (of target schema), it is called a *solution*. We illustrate the above notions with the help of the following example.

Example 1.3.1 (cont. Example 1.2.1 and 1.2.2)**.** Suppose that the conference database needs to be transformed into a database that follows a different relational schema. Let this *target schema* contain the relations: *Author*, *Publication* and *Conference*; and the constraints be the following dependencies:

$$\textit{Author} : IdAuthor \rightarrow IdAuthor\ Name\ Country$$

$$\textit{Conference} : Name\ Year \rightarrow Name\ Year\ BookTitle\ Place$$

$$\textit{Publication} : IdAuthor\ Title \rightarrow IdAuthor\ Title\ BookTitle$$

$$\textit{Publication} : Title \rightarrow Title\ BookTitle$$

$$\textit{Publication}[BookTitle] \subseteq \textit{Conference}[BookTitle]$$

$$\textit{Publication}[IdAuthor] \subseteq \textit{Author}[IdAuthor]$$

The relation *Author* stores the names of authors and their countries. The relation *Conference* stores conference names, the year of realization, the booktitle and the place. The relation *Publication* stores publications, the authors of each publication and the booktitle. Now, the following rules for transforming the conference database into the new schema are listed. We also give the logical definition that we explain latter.

- A talk presented by a researcher is mapped to a publication and the researcher

who authors such a talk is mapped to an author. Formally,

$$\textit{Researcher}(x_1, x_2, x_3, x_4, x_5) \wedge \textit{Talk}(x_1, x_6, x_7) \Rightarrow \exists y_1, y_2.\, \textit{Author}(x_1, x_2, y_1) \wedge \textit{Publication}(x_1, x_6, y_2). \quad (1.3)$$

- A paper presented in a talk that was written by collaborator researcher is mapped to a publication and the collaborator researcher is mapped to an author. Formally,

$$\textit{Collaborator}(x_1, x_2, x_3, x_4) \wedge \textit{Researcher}(x_4, x_5, x_6, x_7, x_8) \Rightarrow \exists y_1, y_2.\, \textit{Author}(x_4, x_5, y_1) \wedge \textit{Publication}(x_1, x_3, y_2). \quad (1.4)$$

- Every conference is mapped to a conference in the new schema. Formally,

$$\textit{Conference}(x_1, x_2, x_3, x_4) \Rightarrow \exists y_1.\, \textit{Conference}(x_2, x_3, y_1, x_4). \quad (1.5)$$

Figure 1.4 shows a *target instance* of the target schema obtained from the application of the set of rules to the source instance (conference database). This target instance includes null values ($\perp_1, \perp_2, \perp_3$) to fill those attribute values that were not extracted from the source instance. We see that this target instance is a solution because the set of functional and inclusion dependencies are satisfied. We denote this data exchange setting by $\mathscr{E}_0$ that is composed of the *source relational schema* of the conference database, the *target relational schema* and the set of rules specified before. □

In the data exchange setting of Example 1.3, the existential variables represent the new fresh values that will be introduced in a tuple. There are cases where these new attribute values depend on the source instance such as transformation or extraction of attribute values or a set of attribute values. The representation of such transformation or extraction is done by the help of function names. If the function creates fresh values that depend on the set of all attributes then the function is called *Skolem function*. We illustrate the use of function names with the next example.

Example 1.3.2. Consider the target instance of Example 1.4 as the source database for a new data exchange setting. The target schema of this data exchange setting contains

Author

Id	*Name*	*Country*
1	*Jose Lozano*	$\perp_1$
2	*Edith Tupac*	$\perp_1$
3	*Steve*	$\perp_1$
4	*Bernard*	$\perp_1$
5	*Lucia*	$\perp_1$

Conference

Name	*Year*	*BookTitle*	*Place*
ICDT	2019	$\perp_2$	*Lille*
ISWC	2019	$\perp_3$	*Chile*
AMW	2015	*Proc. AMW*	*Cuba*

Publication

IdAuthor	*Title*	*BookTitle*
1	*Tutoring Web math platform*	$\perp_2$
1	*Skolemization of prenex formulas*	$\perp_2$
2	*On algebraic connectivity of graphs*	$\perp_3$
3	*On algebraic connectivity of graphs*	$\perp_3$
4	*Tutoring Web math platform*	$\perp_2$
5	*Skolemization of prenex formulas*	$\perp_2$

Figure 1.4: Example of a solution.

only the relation *Book*, which stores a list of books with the attributes: id, title, country key abbreviated by ck and editor. The rule for transformation is that every conference is mapped to a book where:

- id is a number generated by a function whose parameters are name and year of conference;
- title is the booktitle;
- ck is the value that depends on the transformation of place; and
- editor is an invented value that depends on all attributes of conference.

Formally,

$$\exists f_{ed}.\ \textit{Conference}(x_1, x_2, x_3, x_4) \Rightarrow$$
$$\textit{Book}(f_{id}(x_3, x_4), x_3, f_{ck}(x_4), f_{ed}(x_1, x_2, x_3, x_4)), \quad (1.6)$$

where f_{ed} is a Skolem function, f_{id} and f_{ck} are function names whose interpretations are as follows.

- f_{id} performs the hash of the concatenation of name with year; and
- f_{ck} assigns an international country code by the value of the place.

A solution for this new data exchange setting is shown in Figure 1.5. □

Book

Id	*Name*	*Ck*	*Editor*
422f7196b35	$\perp_2$	33	$f_{ed}(\textit{ICDT}, 2019, \perp_2, \textit{Lille})$
9d1a8988af4	$\perp_3$	56	$f_{ed}(\textit{ISWC}, 2019, \perp_3, \textit{Chile})$
fe257fa3467	*Proc. AMW*	53	$f_{ed}(\textit{AMW}, 2015, \textit{Proc. AMW}, \textit{Cuba})$

Figure 1.5: A solution for data exchange setting with function names.

1.3.1 Data exchange setting

A *data exchange setting* is a tuple $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathscr{F})$, where

- $\mathbf{R} = (\mathscr{R}_{\mathbf{R}}, attrs_{\mathbf{R}}, \Sigma^{\mathbf{R}}_{\text{fd}}, \Sigma^{\mathbf{R}}_{\text{ind}})$ is a source relational schema,
- $\mathbf{S} = (\mathscr{R}_{\mathbf{S}}, attrs_{\mathbf{S}}, \Sigma^{\mathbf{S}}_{\text{fd}}, \Sigma^{\mathbf{S}}_{\text{ind}})$ is a target relational schema,
- $\mathscr{F} \subseteq \mathscr{V}_f$ is a set of function names and
- Σ_{st} is a set of *source-to-target tuple generating dependencies* (st-tgds) where each st-tgd σ is of the form

$$\forall \mathbf{x}.\ \varphi(\mathbf{x}) \Rightarrow \exists \mathbf{y}.\ \psi(\mathbf{x}, \mathbf{y}).$$

We distinguish two sub-classes:

- when $\mathscr{F} = \emptyset$, we call it *first-order data exchange* and we write w.l.o.g. $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}})$
- otherwise, we call it *Skolemized data exchange*.

Usually, the set of inclusion dependencies in the target schema are called *target tuple generating dependencies (t-tgds)* because the dependencies are expressed as tgds. We recall that the set of functional dependencies are expressed as egds. Given a data

exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}})$ and a consistent instance I of relational schema $\mathbf{R}$, a *solution* to I w.r.t. $\mathscr{E}$ is any instance J of $\mathbf{S}$ such that $I \cup J$ satisfy the set of dependencies Σ_{st}. By $sol_{\mathscr{E}}(I)$, we denote the set of all solutions to I w.r.t. $\mathscr{E}$.

1.3.2 Chase procedure

The *chase* algorithm was originally used to decide logical implication of first-order constraints [Maier *et al.* 1979]. Here, we use the chase to construct *solutions* to *data exchange*. The chase procedure begins with a given source instance and iteratively constructs the solution by applying the given set of dependencies, which can be *tgds* and *egds*. At each *step* the chase identifies a dependency that is *triggered* i.e., the body of the dependency is satisfied and the head is not, and then applies it, i.e., if the dependency is a tgd, then its application generates new tuples, and if the dependency is an egd, then its application attempts an equation of values, which may result in a *failure* if the values are different constants. In general, the *chase sequence* might be infinite, but if it terminates, then it can end with a failure or give a consistent solution. This chase variant is called *restricted chase*. We recall the notion of chase with the help of the following example.

Example 1.3.3 (cont. Example 1.3.1 and 1.2.2)**.** Recall the conference database I and the set of st-tgds Σ_{st} (rules in Example 1.3.1). The procedure initializes with an empty instance J of the target schema. In Figure 1.6 we observe that the st-tgd (1.3) is triggered because the body of the dependency matches values in the instance I and the head of the dependency does not match values in instance J. The application of this dependency generates fresh values for matching the head of the dependency in J. By triggering the rules in Example 1.3.1, we get the following *chase sequence*.

- Dependency (1.3) is triggered by two tuples

  $$\textit{Researcher}(1, \textit{Jose Lozano}, \textit{j@s.fr}, \textit{Math}, \textit{ALG}) \text{ and } \textit{Talk}(1, 1, \textit{Tutor}\ldots)$$

  and applying this dependency results in adding the tuples

  $$\textit{Author}(1, \textit{Jose Lozano}, \perp_4), \textit{Publication}(1, \textit{Tutor}\ldots, \perp_5).$$

$$\textit{Researcher}(x_1, x_2, x_3, x_4, x_5) \wedge \textit{Talk}(x_1, x_6, x_7) \Rightarrow \exists y_1, y_2.\, \textit{Author}(x_1, x_2, y_1) \wedge \textit{Publication}(x_1, x_6, y_2)$$

Researcher

Idr	*Name*	*Email*	*Expertise*	*Team*
1	*Jose Lozano*	*j@s.fr*	*Math*	*ALG*
3	*Steve*	*s@s.fr*	*Algebra*	*ALG*

Talk

Idr	*Idc*	*Title*
1	1	*Tutoring Web math platform*
1	1	*Skolemization of prenex formulas*
3	2	*On algebraic connectivity of graphs*

Researcher					*Talk*		
x_1	x_2	x_3	x_4	x_5	x_1	x_6	x_7
1	*Jose Lozano*	*j@s.fr*	*Math*	*ALG*	1	1	*Tutor...*

Author		
x_1	x_2	y_1
1	*Jose Lozano*	$\perp_4$

Publication		
x_1	x_6	y_2
1	*Tutor...*	$\perp_5$

Figure 1.6: Matching of the body and the application of the head of the st-tgd 1.3.

- Dependency (1.3) is triggered by two tuples

  $$\textit{Researcher}(1, \textit{Jose Lozano}, \textit{j@s.fr}, \textit{Math}, \textit{ALG}) \text{ and } \textit{Talk}(1, 1, \textit{Skol...})$$

  and applying this dependency results in adding only the tuple

  $$\textit{Publication}(1, \textit{Skol...}, \perp_6).$$

  Recall that $\textit{Author}(1, \textit{Jose Lozano}, \perp_4)$ is already there. We observe that the application of a tgd not always generates the whole head.

- Analogously, other dependencies are triggered and applied until there is no de-

pendency that is triggered. Those dependencies are (1.3), which is triggered one more time, (1.4), which is triggered twice, and (1.5), which is triggered twice.

At the end of the chase sequence, we have the target instance, which we denote by J_0, seen in Figure 1.7. This instance is called a *pre-solution*. We point out that the chase with the set of st-tgds always terminates.

Author

Id	*Name*	*Country*
1	*Jose Lozano*	$\perp_4$
2	*Edith Tupac*	$\perp_{10}$
3	*Steve*	$\perp_{11}$
4	*Bernard*	$\perp_{12}$

Conference

Name	*Year*	*BookTitle*	*Place*
ICDT	2019	$\perp_{13}$	Peru
ISWC	2019	$\perp_{14}$	Chile

Publication

IdAuthor	*Title*	*BookTitle*
1	*Tutoring Web math platform*	$\perp_5$
1	*Skolemization of prenex formulas*	$\perp_6$
2	*On algebraic connectivity of graphs*	$\perp_7$
3	*On algebraic connectivity of graphs*	$\perp_8$
4	*Tutoring Web math platform*	$\perp_9$

Figure 1.7: A pre-solution.

Now, we proceed with the set of target constraints where the chase is used to enforce the satisfaction of these constraints on the instance J_0. We take those constraints as set of dependencies (tgds and egds).

$$\textit{Author}(x_1, x_2, x_3) \wedge \textit{Author}(x_1, y_2, y_3) \Rightarrow x_2 = y_2 \wedge x_3 = y_3, \tag{1.7}$$

$$\textit{Conference}(x_1, x_2, x_3, x_4) \wedge \textit{Conference}(x_1, x_2, y_3, y_4) \Rightarrow x_3 = y_3 \wedge x_4 = y_4, \tag{1.8}$$

$$\textit{Publication}(x_1, x_2, x_3) \wedge \textit{Publication}(x_1, x_2, y_3) \Rightarrow x_3 = y_3, \tag{1.9}$$

$$\textit{Publication}(x_1, x_2, x_3) \wedge \textit{Publication}(x_4, x_2, y_3) \Rightarrow x_3 = y_3, \tag{1.10}$$

$$\textit{Publication}(x_1, x_2, x_3) \Rightarrow \exists y_1, y_2, y_3.\ \textit{Conference}(y_1, y_2, x_3, y_3), \tag{1.11}$$

$$\textit{Publication}(x_1, x_2, x_3) \Rightarrow \exists y_1, y_2.\ \textit{Author}(x_1, y_1, y_2). \tag{1.12}$$

The chase sequence is as follows.

- Dependency (1.10) is triggered by two tuples

$$\textit{Publication}(1, \textit{Tutor}\ldots, \perp_5) \text{ and } \textit{Publication}(4, \textit{Tutor}\ldots, \perp_9)$$

  and applying this dependency results in equating the values

$$\perp_9 = \perp_5.$$

- This dependency also triggers for publication of title *On alg...* and equates

$$\perp_7 = \perp_8.$$

- Dependency (1.11) is triggered three times adding new tuples.

The chase sequence ends with the following solution seen in Figure 1.9. □

The chase sequence with these target dependencies and the rules of Example 1.3.1 is *finite*. However, the chase sequence can finish with a *failure* or even be *infinite*. A failure is when in the application of the egd, the two values to be equated are constants and not equal. In the following example, we illustrate the notion of failure.

Example 1.3.4. Suppose we have the table conference in Figure 1.8 of Conference database. During the chase, dependency (1.5) is triggered by this tuple adding

Conference

Idc	*Name*	*Year*	*Place*
1	*ICDT*	2019	*Peru*
2	*ISWC*	2019	*Chile*
3	*ICDT*	2019	*Chile*

Figure 1.8: An instance of conference table.

$$\textit{Conference}(\textit{ICDT}, 2019, \perp, \textit{Chile}).$$

Dependency (1.8) is triggered by the last tuple added and the tuple

$$\textit{Conference}(\textit{ICDT}, 2019, \perp, \textit{Peru}).$$

But the application cannot equate

$$\mathit{Peru} = \mathit{Chile}.$$

Thus, the chase ends with a failure. □

A chase sequence is infinite when the application of a tgd always triggers another tgd. In the following example, we illustrate an infinite chase.

Example 1.3.5. Consider the source schema $\mathbf{R}_1$ with relation $\mathit{Worker}(\underline{id}, name, age)$, and the target schema $\mathbf{R}_2$ with two relations $\mathit{Emp}(\underline{id}, name)$ and $\mathit{Sup}(super\_id, emp\_id)$. The target schema also contains the following tgd constraints.

$$\begin{aligned} &\mathit{Sup}(x_1, x_2) \Rightarrow \exists y.\ \mathit{Emp}(x_1, y), \\ &\mathit{Sup}(x_1, x_2) \Rightarrow \exists y.\ \mathit{Emp}(x_2, y), && (1.13) \\ &\mathit{Emp}(x_1, x_2) \Rightarrow \exists y.\ \mathit{Sup}(x_1, y). && (1.14) \end{aligned}$$

These constraints impose that every employee has a supervisor and the value of emp_id needs to exist in the database as an employee and the supervisor also is an employee. The st-tgd for exchanging data is

$$\mathit{Worker}(x_1, x_2, x_3) \Rightarrow \mathit{Emp}(x_1, x_2).$$

Take the following source instance $\mathit{Worker}(1, \mathit{Jose}, 30)$. After applying the st-tgd we have an instance J_1 with only one tuple $\mathit{Emp}(1, \mathit{Jose})$ and we observe that the tgd (1.14) is triggered which results in adding the tuple $\mathit{Sup}(\bot_1, 1)$. Then this triggers the tgd (1.13) which results in adding $\mathit{Emp}(\bot_1, \bot_2)$. Then, the tgd (1.14) is triggered again, which results in adding the tuple $\mathit{Sup}(\bot_3, \bot_1)$ that causes tgd (1.13) to be triggered again adding the tuple $\mathit{Emp}(\bot_3, \bot_4)$. Such result triggers the tgd (1.14) already triggered before adding $\mathit{Sup}(\bot_5, \bot_3)$. This tgd (1.14) together with the tgd (1.13) are triggered ad infinitum.

Author

Id	*Name*	*Country*
1	*Jose Lozano*	$\perp_4$
2	*Edith Tupac*	$\perp_{10}$
3	*Steve*	$\perp_{11}$
4	*Bernard*	$\perp_{12}$

Conference

Name	*Year*	*BookTitle*	*Place*
ICDT	2019	$\perp_{13}$	*Peru*
ISWC	2019	$\perp_{14}$	*Chile*
$\perp_{15}$	$\perp_{16}$	$\perp_5$	$\perp_{17}$
$\perp_{18}$	$\perp_{19}$	$\perp_6$	$\perp_{20}$
$\perp_{21}$	$\perp_{22}$	$\perp_7$	$\perp_{23}$

Publication

IdAuthor	*Title*	*BookTitle*
1	*Tutoring Web math platform*	$\perp_5$
1	*Skolemization of prenex formulas*	$\perp_6$
2	*On algebraic connectivity of graphs*	$\perp_7$
3	*On algebraic connectivity of graphs*	$\perp_7$
4	*Tutoring Web math platform*	$\perp_5$

Figure 1.9: A Universal Solution.

Homomorphism

We define triggering of a dependency with the classical notion of *homomorphism*. First, we define a function called *substitution* as follows $h : \mathscr{V}_a \cup \mathsf{Dom} \rightarrow \mathscr{V}_a \cup \mathsf{Dom}$. This function is different from the identity function on the set of null values and variables, which we denote by $adom(h)$. Furthermore, h assigns a value in Dom to every element in $adom(h)$.

Now, a *homomorphism* $h : I_1 \rightarrow I_2$ between two relational structures I_1, I_2 of the same relational signature $\mathscr{R}$ is a substitution from $adom(I_1)$ to $adom(I_2)$ that

- preserves the constant values i.e., $h(a) = a$ whenever $a \in \mathsf{Const}$, and
- for every $R \in \mathscr{R}$ and every $(a_1, \ldots, a_n) \in I_1(R)$ we have $(h(a_1), \ldots, h(a_n)) \in I_2(R)$ and n is the arity of R.

A homomorphism h' *extends* a homomorphism h, written $h \subseteq h'$, if $adom(h) \subseteq adom(h')$ and $h'(x) = h(x)$ for all $x \in adom(h)$.

We define a mapping h^F that will be used to define the notion of homomorphism from a formula into an instance. We use the notion of this homomorphism for chase purposes. Let $\mathscr{F}$ be a function signature and F be an interpretation of $\mathscr{F}$. For a term t over $\mathscr{F}$ and a substitution $h : \mathscr{V}_a \cup \mathsf{Dom} \to \mathscr{V}_a \cup \mathsf{Dom}$, we define $h^F(t)$ as:

$$h^F(t) = \begin{cases} h(x) & \text{if } t = x \in \mathscr{V}_a \\ a & \text{if } t = a \in \mathsf{Dom} \\ f(h^F(t')) & \text{if } t = f(t') \text{ is a function term.} \end{cases}$$

The mapping h^F is extended on atoms and conjunctions of atoms as expected:

$$h^F(R(t,\ldots,t)) = R(h^F(t,\ldots,t)) \text{ and } h^F(\bigwedge_{i\in 1..k} R_i(t_i,\ldots,t_i)) = \bigwedge_{i\in 1..k} h^F(R_i(t_i,\ldots,t_i)).$$

Note that if the argument of h^F does not contain function terms, the interpretation F is irrelevant so we allow to omit the F superscript and write e.g. $h(t,\ldots,t)$ instead of $h^F(t,\ldots,t)$. A *homomorphism* $h : \varphi \to I \cup F$ between the conjunction of atoms φ over signature $\mathscr{R} \cup \mathscr{F}$ to an instance union the interpretation of function symbols, is a mapping from $fvars_{\mathscr{R}\cup\mathscr{F}}(\varphi)$ to Dom s.t. for every atom $R(t,\ldots,t)$ in φ it holds that $R(h^F(t,\ldots,t)) \in I$. Remark that if φ does not contain function terms, then F in the above definition is irrelevant and we write $h : \varphi \to I$ instead of $h : \varphi \to I \cup F$ and $h(t,\ldots,t)$ instead of $h^F(t,\ldots,t)$.

Chase

We define formally the *chase* procedure for tgds and egds. Let I be a database instance of a relational schema $\mathbf{R} = (\mathscr{R}, attrs, \Sigma_{\text{fd}}, \Sigma_{\text{ind}})$. Let σ be a tgd of the form

$$\forall \mathbf{x}.\ \varphi \Rightarrow \exists \mathbf{y}.\ \psi$$

where φ and ψ are over $\mathscr{R}$. We say that σ is *triggered* in I by h if

- $adom(h) = \mathbf{x}$,
- $h(\varphi) \subseteq I$, 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

$$adom(h') = \mathbf{x} \cup \mathbf{y} \text{ and } I' = I \cup h'(\psi).$$

Next, let σ be an egd of the form

$$\forall \mathbf{x}.\ \varphi \Rightarrow x_i = x_j$$

where $\{x_i, x_j\} \subseteq \mathbf{x}$. We say that σ is *triggered* in I by h if

- $adom(h) = \mathbf{x}$,
- $h(\varphi) \subseteq I$, and
- $h(x_i) \neq h(x_j)$.

It has a successful *execution* h' yielding I', in symbols $I \xrightarrow{\sigma,h'} I'$, if there is a homomorphism h' such that $adom(h') = h(adom(h)) \cap \text{Null}$ i.e., h' assigns values to the null values used by h, and we have $h'(h(x_i)) = h'(h(x_j))$ while the other null values remain the same; and $I' = h'(I)$. If σ is triggered in I by h but does not have a successful execution, we say that it *fails*, in symbols $I \xrightarrow{\sigma,h} \perp$.

Now, we define the chase for a st-tgd. Let $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma)$ be a data exchange setting. A st-tgd $\sigma \in \Sigma$ is a tgd with the difference that φ is over the source signature $\mathscr{R}_{\mathbf{R}}$ and ψ is over the target signature $\mathscr{R}_{\mathbf{S}}$. Here the chase takes an instance $I = I_{\mathbf{R}} \cup I_{\mathbf{S}}$, where $I_{\mathbf{R}}$ is the source instance and $I_{\mathbf{S}}$ is the target instance. We say that σ is *triggered* in I by h if

- $adom(h) = \mathbf{x}$,
- $h(\varphi) \subseteq I_{\mathbf{R}}$, and
- there is no extension h' of h such that $h'(\psi) \subseteq I_{\mathbf{S}}$.

It has a successful *execution* h' yielding I', in symbols $I \xrightarrow{\sigma,h'} I'$, if h' is an extension of h such that

$$adom(h') = \mathbf{x} \cup \mathbf{y} \text{ and } I' = I \cup h'(\psi).$$

Each time a dependency σ is triggered in I by a homomorphism h is called a *chase step*. Now, given an instance I and a set Σ_{st} of st-tgds together with a set of t-tgds Σ_{ind} and egds Σ_{fd}, a *chase sequence* of I with $\Sigma_{\text{st}} \cup \Sigma_{\text{ind}} \cup \Sigma_{\text{fd}}$ is a possibly infinite sequence of chase steps $I_0 \xrightarrow{\sigma_0,h_0} I_1 \xrightarrow{\sigma_1,h_1} I_2 \ldots$, where $I_0 = I$ and $\sigma_i \in \Sigma_{\text{st}} \cup \Sigma_{\text{fd}} \cup \Sigma_{\text{ind}}$ for all i. A *terminating* chase sequence ends with a failure or an instance that triggers no dependency in $\Sigma_{\text{st}} \cup \Sigma_{\text{ind}} \cup \Sigma_{\text{fd}}$. Since at each step we non deterministically trigger a tgd and egd, then the chase returns different instances. In general, given a relational schema $\mathbf{R}$, a set of set of tgds and egds Σ, and a target schema $\mathbf{S}$, a chase is a relation $chase \subseteq Inst(\mathbf{R}) \times 2^{\Sigma} \times Inst(\mathbf{S})$. But, if Σ is composed only of egds or tgds that do not have existential quantifiers then the chase is a function because returns only one target instance such that $J = chase(I, \Sigma)$.

1.3.3 Universal solution

Given a data exchange setting and a source instance, there might be no solution or possibly infinite number of solutions, and a considerable amount of work has focused on finding *universal solutions* that represent the entire space of solutions. Intuitively, a universal solution is the most general solution that is included in all specific solutions. As we shall see later on, universal solutions are useful for computing *answers* to a given query over all solutions. We illustrate the significance of being a particular and general solution by their comparison in the following example.

Example 1.3.6. Recall the solution in Figure 1.9 denoted by $\mathscr{J}$ and the solution in Figure 1.4 denoted by J_1. We make the following observations.

- J_1 contains data values not present in the source instance e.g., author *Lucia* and the conference name *AMW*. Also this solution contains facts that cannot be derived from source instance using st-tgds. For instance, the publication with title *On algebraic connectivity of graphs* is in the conference *ISWC*.
- In J_1, all authors are from the same country.
- On the other hand, $\mathscr{J}$ contains only data values that are present in the source instance and null values invented to satisfy target constraints. Also, $\mathscr{J}$ contains only facts that are derived from source instance.

Comparing these two solutions, we observe that J_1 has been constructed with some additional information that makes it more specific solution than $\mathscr{J}$. Furthermore, we see all the information of $\mathscr{J}$ can be found in J_1, but not all the information of J_1 can be found in $\mathscr{J}$. □

We use homomorphisms to compare solutions: we say that $\mathscr{J}$ *subsumes* J' if there is a homomorphism from $\mathscr{J}$ to J'.

Example 1.3.7. We define a homomorphism h from $\mathscr{J}$ to J_1 such that $h(\mathscr{J}) \subseteq J_1$ as follows.

$h(\bot_4) = \bot_1$ $h(\bot_{10}) = \bot_1$ $h(\bot_{11}) = \bot_1$ $h(\bot_{12}) = \bot_1$
$h(\bot_{15}) = \textit{ICDT}$ $h(\bot_{18}) = \textit{ICDT}$ $h(\bot_{21}) = \textit{ISWC}$ $h(\bot_{16}) = 2019$
$h(\bot_{19}) = 2019$ $h(\bot_{22}) = 2019$ $h(\bot_5) = \bot_2$ $h(\bot_6) = \bot_2$
$h(\bot_{13}) = \bot_2$ $h(\bot_7) = \bot_3$ $h(\bot_{14}) = \bot_3$ $h(\bot_{17}) = \textit{Peru}$
$h(\bot_{20}) = \textit{Peru}$ $h(\bot_{23}) = \textit{Chile}$

We start by mapping all tuples of *Conference* from $\mathscr{J}$ to J_1 using h previously defined such that the image of every tuple of $\mathscr{J}$ can be found in J_1. For instance, the tuple *Conference*(*ICDT*, 2019, $\bot_{13}$, *Peru*) is mapped to *Conference*(*ICDT*, 2019, $\bot_2$, *Peru*) by $h(\bot_{13}) = \bot_2$. Then all tuples of *Author* from $\mathscr{J}$ are mapped to J_1. For instance, the tuple *Author*(3, *Steve*, $\bot_{11}$) is mapped to *Author*(3, *Steve*, $\bot_1$) by $h(\bot_{11}) = \bot_1$. Finally all tuples of *Publication* from $\mathscr{J}$ are mapped to J_1. For instance, the tuple *Publication*(1, *Tutoring...*, $\bot_5$) is mapped to *Publication*(1, *Tutoring...*, $\bot_2$) by applying $h(\bot_5) = \bot_2$. These mappings seen in Figure 1.10 prove that J_1 is subsumed by $\mathscr{J}$. □

A universal solution subsumes every solution from the set of solutions for a given instance w.r.t. a data exchange setting. Formally, we define a universal solution as follows.

Definition 1.3.8. Given a data exchange setting $\mathscr{E}$ and a source instance I, a solution $\mathscr{J} \in sol_{\mathscr{E}}(I)$ is *universal* iff for every solution J' to I w.r.t. $\mathscr{E}$ there is a homomorphism h from $\mathscr{J}$ to J'.

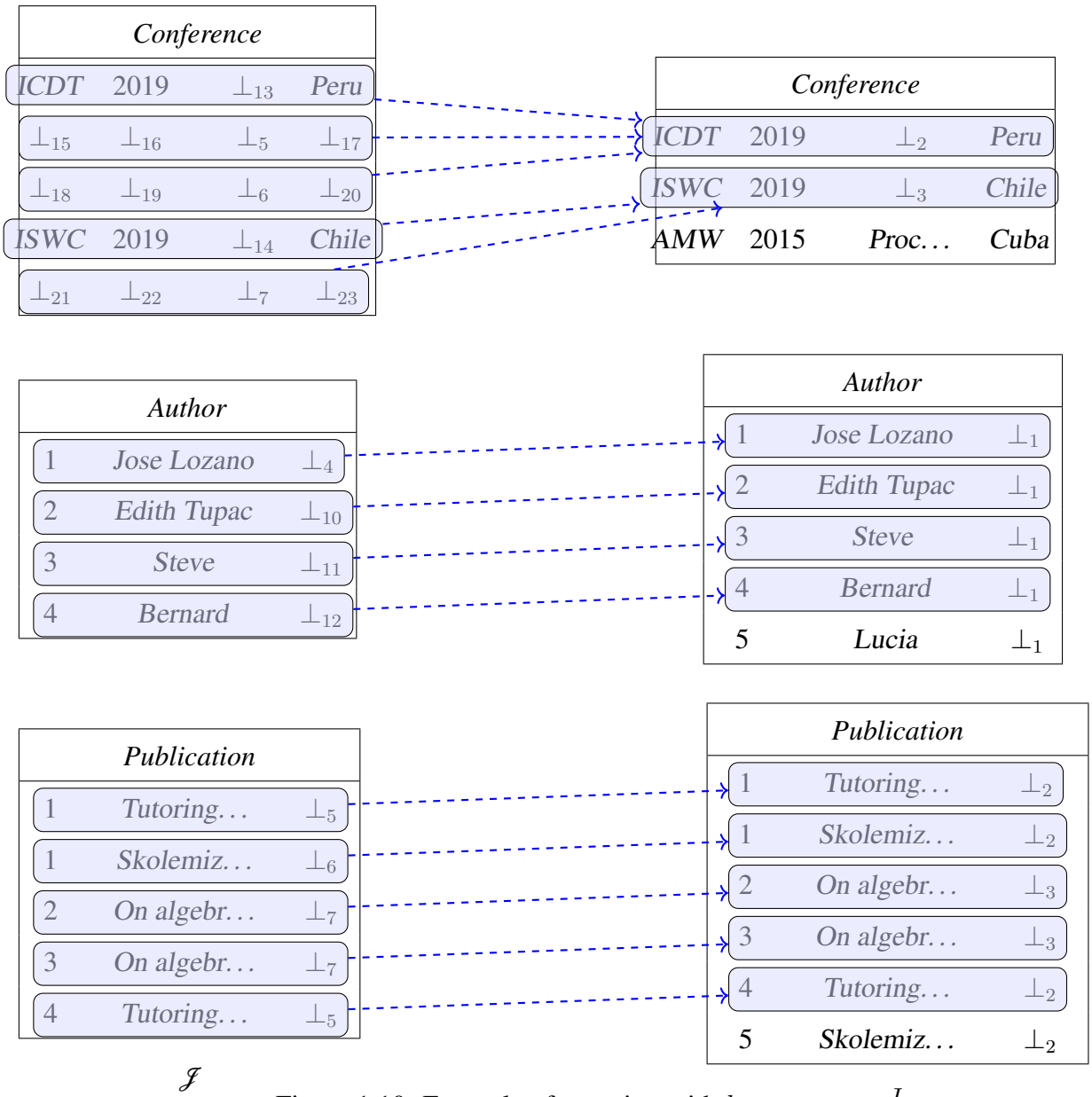


Figure 1.10: Example of mapping with h.

Existence of universal solution

As we have mentioned before, there are cases where chase sequence is infinite for a given instance and a data exchange setting. However in such cases, a solution may still exist. In the Example 1.3.5 we have presented an infinite chase sequence for a setting and a source instance, yet in Figure 1.11 we present a number of solutions. It has proven in [Fagin *et al.* 2005a] that infinite chase implies there is no finite universal solution. In

J_1

Emp		*Sup*	
id	*name*	*super_id*	*emp_id*
1	Jose	1	1

J_2

Emp		*Sup*	
id	*name*	*super_id*	*emp_id*
1	Jose	$\perp_1$	1
$\perp_1$	$\perp_2$	1	$\perp_1$

J_3

Emp		*Sup*	
id	*name*	*super_id*	*emp_id*
1	Jose	$\perp_1$	1
$\perp_1$	$\perp_2$	$\perp_3$	$\perp_1$
$\perp_3$	$\perp_4$	1	$\perp_1$

Figure 1.11: Solutions for Example 1.3.5.

Figure 1.11, we view solutions as graphs where nodes are the data values and edges are the relation names that contain the data values. The corresponding graph of each solution is seen in Figure 1.12. In general, for any $i \in \mathbb{N}$, we can construct J_i that corresponds to a cycle of length i and uses $2 * i - 2$ null values. We observe that if a solution J exists and there is a homomorphism from J to J_i then J needs $2 * i - 2$ different null values. So if a universal solution exists then a homomorpshim exists from J to J_i for any i that belongs that set of natural numbers, which is an infinite set. Therefore, J would have infinite null values. In fact, chase is attempting to construct an instance that is a path that never ends.

1.3.4 Certain query answering

Next, we consider the problem of query answering. We know the definition of an answer to a query over a single database. Here the challenge is to define what is an answer to a query over a set of instances. The framework *possible-world semantics* allows to answer this question.

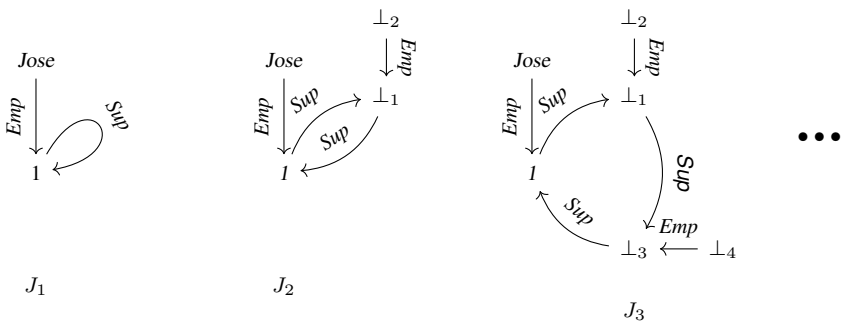


Figure 1.12: Graph representation of the set of solutions in Figure 1.11.

Possible-world semantics has been introduced to handle uncertain databases. An uncertain database is a database with named null values and it represents a set of ground databases where null values have been replaced with concrete values. A *certain* answer to a query over an uncertain database is an answer present in every ground database of the uncertain database.

This framework serves as inspiration for defining *certain answers* to a query over a set of solutions. Formally, we define certain answers as follows.

Definition 1.3.9. Given a data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}})$, a source instance I and a query Q over the target schema $\mathbf{S}$, an answer to query Q with respect to I is *certain* if this answer is an answer to query Q in every solution to I w.r.t. $\mathscr{E}$. □

In case of Boolean queries, $true$ is the certain answer to Boolean query Q if $true$ is the answer to query Q in every solution to I w.r.t. $\mathscr{E}$. We illustrate answers that are certain and not certain to a query in the following example.

Example 1.3.10. Recall the solution $\mathscr{J}$ in Figure 1.9 and the solution J_1 in Figure 1.4 for the source instance I in Figure 1.1 w.r.t. data exchange setting $\mathscr{E}_0$ in Example 1.3.1. Consider the query $Q_1(y) = \exists x_1, x_2, x_3.$ *Conference*(y, x_1, x_2, x_3) asking what are the names of the conferences. In fact, we see that the certain answers are (ICDT) and (ISWC). An example of not certain answer is *AMW*.

Query \ Solution	J_1	$\mathscr{J}$
Q_1	$\{(\mathit{ICDT}), (\mathit{ISWC}), (\mathit{AMW})\}$	$\{(\mathit{ICDT}), (\mathit{ISWC}),$ $(\perp_{15}), (\perp_{18}), (\perp_{21})\}$

Figure 1.13: Answers to queries Q_1 in solutions J_1 and $\mathscr{J}$.

Consider the following Boolean queries:

(Q_2) $\exists x_1, x_2, x_3, x_4, x_5.$ *Author*$(x_1,$ *Steve*$, x_2) \wedge$ *Conference*$($*ISWC*$, 2019, x_3, x_4) \wedge$ *Publication*(x_1, x_5, x_3) asking if *Steve* has a publication in an *ISWC* conference, and

(Q_3) $\exists x_1, x_2, x_3, x_4, x_5, x_6.$ *Author*$(x_1,$ *Jose Lozano*$, x_2) \wedge$ *Author*$(x_3,$ *'Bernard*$, x_4) \wedge$ *Publication*$(x_1, x_5, x_6) \wedge$ *Publication*(x_3, x_5, x_6) asking if authors *Jose Lozano* and *Bernard* have co-authored the same publication.

	J_1	$\mathscr{J}$
Q_2	$true$	$false$
Q_3	$true$	$true$

Figure 1.14: Answers to Boolean queries Q_2 and Q_3 in solutions J_1 and $\mathscr{J}$.

Figure 1.14 shows that $true$ is not certain answer to query Q_2. □

A straightforward attempt at computing certain answers is to evaluate the query in all solutions and to return the intersection of all answers corresponding to each solution. However this attempt is not feasible when the number of solutions is infinite. An alternative solution, which is proposed by Fagin et.al [Fagin *et al.* 2005a], is the evaluation of the query over a *universal solution*.

Example 1.3.11. Consider the source instance I in Figure 1.1 and the data exchange setting $\mathscr{E}_0$ in Example 1.3.1. Take $\mathscr{J}$ that is a universal solution and the set of answers to Q_1. Tuples that contain null values are dropped from the result obtaining that (*ICDT*) and (*ISWC*) are certain answers. Also for the Boolean query Q_3 take $\mathscr{J}$ and $true$ is the certain answer. □

Computing certain answers is defined as follows. We fix a source instance I, data exchange setting $\mathscr{E}$ and a query Q, we denote the set of certain answers for a query φ of I w.r.t. $\mathscr{E}$ by $cert_{\mathscr{E}}(Q, I)$. Formally, certain answers is defined by the intersection of the set of answers to query Q over each solution of the set of solutions to I w.r.t. $\mathscr{E}$, in symbols

$$cert_{\mathscr{E}}(Q, I) = \bigcap \{ QA(Q, J) \mid J \in sol_{\mathscr{E}}(I) \}.$$

1.3.5 Consistency

A data exchange setting is inconsistent if there is a source relational instance for which the set of solutions is empty. If a data exchange setting admits solutions for every source instance then it is called *consistent*. We illustrate a setting that is not consistent and how it becomes consistent with another set of source constraints.

Example 1.3.12. Recall the data exchange setting in Example 1.3.1. This setting is not consistent because there is no solution for the source instance in Figure 1.15a. Indeed,

Conference

Idc	*Name*	*Year*	*Place*
1	*ICDT*	2019	*Peru*
2	*ISWC*	2019	*Chile*
3	*ICDT*	2019	*Chile*

(a) Source instance.

$\Rightarrow$

Conference

Name	*Year*	*BookTitle*	*Place*
ICDT	2019	$\perp_{27}$	*Peru*
ISWC	2019	$\perp_{28}$	*Chile*
ICDT	2019	$\perp_{29}$	*Chile*

(b) Target instance not satisfying schema.

Figure 1.15: Example of consistency problem.

after applying the set of st-tgds we obtain the instance in Figure 1.15b, where we have two tuples containing the same name and year but with different places. This violates the fd

$$\textit{Conference} : Name\ Year \rightarrow BookTitle\ Place.$$

Since any solution must contains tuples from Figure 1.15b module renaming of the null values, no solution can exist. Therefore the setting is not consistent.

This fd fails because in the source instance of conference database the schema allows to have conferences with the same name but different places. We can prevent it by making the set of attributes *Name, Year* a secondary key, i.e, adding the following functional dependency in the source schema:

$$\textit{Conference} : Name\ Year \rightarrow Place.$$

Source instances that contain same conference name and different places will be removed for consideration, and in fact there is a solution for every consistent source instance. □

Checking consistency is defined as a decision problem that takes as input a data exchange setting and outputs if the setting is consistent or not. Formally, we define consistency as follows.

Definition 1.3.13. A data exchange setting $\mathscr{E}$ is *consistent* if every consistent source instance admits a solution.

1.4 Resource description framework

We present the well-known framework for publishing data on the web that is called *Resource Description Framework (RDF)*. The nodes of RDF is composed of *resources* and simple literal values. A resource is anything that is recognized by a unique identifier called International Resource Identifier (IRI). When an IRI is an anonymous node with no IRI, the resource is called a *blank node*. Properly, RDF dataset is a set of *triples* of the form *subject predicate object*. This form of triple is also called a *statement*. A natural way to see the set of triples is as a *graph* where subjects and objects are *nodes* and the triples themselves are the *labeled edges*. There are different formats for presenting RDF, among those we focus on *N-Triples* and *Turtle*. In the following example, we illustrate these notions.

Example 1.4.1. Suppose the information on math researchers are required to be accessible and identifiable through the Web. The RDF is a model that allows to represent this information with IRIs, literals and blank nodes. An RDF model can be expressed in different formats. We illustrate the same RDF model for math researchers in the formats of N-Triples and Turtle in Figures 1.16 and 1.18, respectively.

Now, we observe in Figure 1.16 the statement that represents the information of the age of researcher *"Jose Lozano"*. We identify *three* elements in this *statement*, which are :

- *subject* that identifies the resource we are describing such as

https://www.univ−lille.fr/jlozano,

```
<https://www.univ-lille.fr/jlozano>
   <http://xmlns.com/foaf/0.1/name> "Jose Lozano".
<https://www.univ-lille.fr/jlozano>
   <http://example.com/ns#masters> <http://dbpedia.org/resource/
   ↪ Mathematics>.
<https://www.univ-lille.fr/jlozano>
   <http://example.com/ns#masters> <http://dbpedia.org/resource/Logic
   ↪ >.
<https://www.univ-lille.fr/jlozano>
   <http://xmlns.com/foaf/0.1/mbox> "j@s.fr".
<https://www.univ-lille.fr/jlozano>
   <http://xmlns.com/foaf/0.1/age> "29".
<https://www.univ-lille.fr/jlozano>
   <http://example.com/ns#worksIn> <https://www.univ-lille.fr/>;
<https://www.univ-lille.fr/jlozano>
   <http://xmlns.com/foaf/0.1/homepage> _:b.
```

Figure 1.16: Set of statements in N-Triples format.

prefix	*namespace*
ex:	http://example.com/ns#
univ:	http://ex.universities/
foaf:	http://xmlns.com/foaf/0.1/
xsd:	http://www.w3.org/2001/XMLSchema#
dbp:	http://dbpedia.org/resource/
dblp:	http://dblp.uni − trier.de/rdf/schema−2015−01−26#
ulille:	https://www.univ−lille.fr/
sh:	http://www.w3.org/ns/shacl#

Table 1.1: RDF prefixes.

- *predicate* that identifies the property of the resource such as

  http://xmlns.com/foaf/0.1/age,

- *object* that identifies the value of the property being another resource or literal value such as *"29"*.

RDF specifies restrictions on the subject and predicate of a triple: the subject cannot be a literal, and the predicate must be an IRI.

As we have seen in format N-Triples, the writing of triples is too verbose. Due to this fact, a set of *prefixes* is used to shorten the writing of IRIs. A prefix is a short name concatenated with the symbol ":". Every prefix is associated to a unique IRI called *namespace*. A namespace is contained in a full IRI and it is usually shorten than the full

IRI. A set of IRIs can share the same namespace. For instance, one namespace in the RDF model of math researchers is https://www.univ−lille.fr. We define the prefix ulille: to be associated to this namespace. Then, the writing of IRIs are shortened as seen in Turtle format (Figure 1.18). For instance, the full IRI https://www.univ−lille.fr/jlozano is rewritten as ulille:jlozano. The set of prefixes used in this manuscript is shown in Table 1.1.

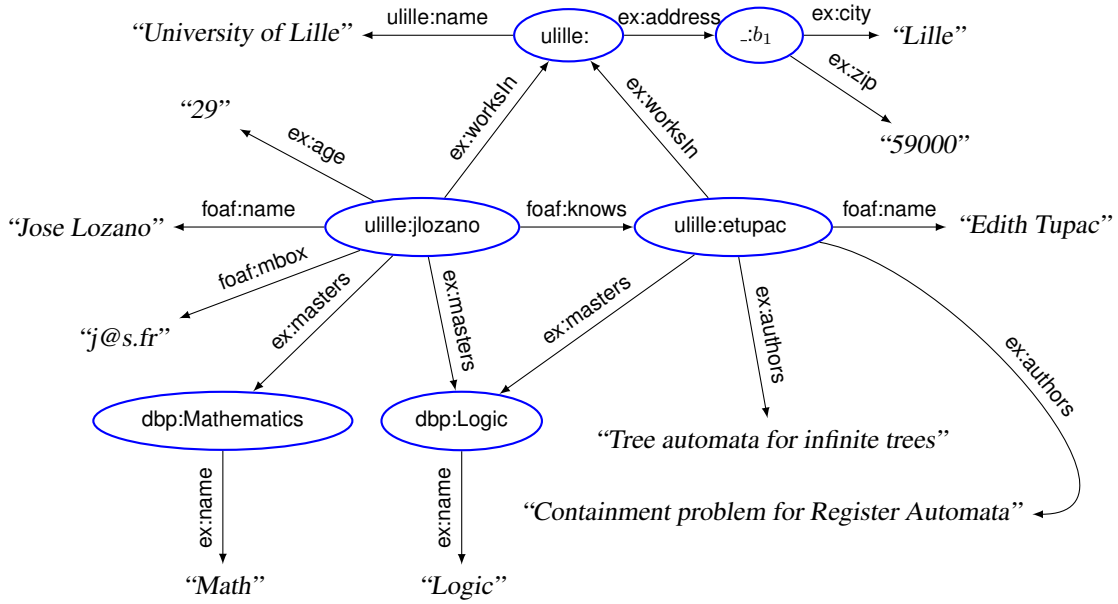


Figure 1.17: An RDF graph where prefixes are in Table1.1.

```
ulille:jlozano
    foaf:name "Jose Lozano";
    ex:masters dbp:Mathematics , dbp:Logic;
    foaf:mbox "j@s.fr" ;
    foaf:age "29";
    ex:worksIn ulille:;
        foaf:homepage _:b.
```

Figure 1.18: Set of statements written in Turtle.

Now, we consider an RDF graph in Figure 1.17 that describes the same researcher *"Jose Lozano"* with the same information in Figure 1.16 and also describes his colleague *"Edith Tupac"* with the following information: she works also in the *"University of Lille"* and has authored two papers: *"Tree automata for infinite trees"* and *"Containment problem for Register Automata"*. The IRI node is shown with an oval and the literal value within quotes. We point out that _:b_1 is a blank node shown with an oval

although is not an IRI.

```
ulille:etupac
   foaf:name "Edith Tupac";
   ex:masters dbp:Logic;
   ex:worksIn ulille:;
   ex:authors "Tree automata for infinite trees" ,
              "Containment problem for Register Automata".
```

Figure 1.19: Serialization of a neighborhood of node ulille:etupac in Turtle.

The outgoing edges of the IRI node ulille:etupac is serialized in Turtle format in Figure 1.19 where we make the following observations:

- when the subject is the same, the set of predicate-object is grouped into a list separating each of them by ";" such as for the subject ulille:etupac; and
- when the subject and predicate are the same, the objects are grouped into a list separating each of them by "," such as for ex:authors.

Given a blank node that has one or more outgoing edges, Turtle simplifies the serialization of this information by writing the outgoing edges inside brackets []. The use of brackets in Turtle represents that a blank node is the subject of the set of predicate objects contained in the brackets. For instance, we serialize the outgoing edges of ulille: in Figure1.20 where one of the target nodes is a blank node. □

```
ulille:
    ulille:name "University of Lille";
    ex:address [
        ex:city "Lille";
        ex:zip "59000"].
```

Figure 1.20: Serialization of the neighborhood of node ulille: in Turtle.

1.4.1 RDF graph

An *RDF graph* (or simply a *graph*) is a finite set G of triples such that

$$G \subseteq (\mathsf{Iri} \cup \mathsf{Blank}) \times \mathsf{Iri} \times (\mathsf{Iri} \cup \mathsf{Blank} \cup \mathsf{Lit}).$$

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.

1.4.2 Logic formalization

The relational signature for a graph is $\mathscr{G} = \{Triple\}$ where $Triple$ a relation name of arity three. For a given graph G, we define the relational structure M_G over $\mathscr{G}$ as $Triple^{M_G} = G$.

1.5 Schemas for RDF graphs

We present the notion of schema based on shapes for RDF graphs used throughout this manuscript. A schema for RDF graph, like a schema for a relational database, defines a structure that a graph should follow and establishes a set of constraints that ensure the integrity of its data. The structure of a graph is given by a set of *types* that are associated with a node and each type of node comes with a set of constraints. These constraints defined in a type ensure that the number of outgoing edges of a node assigned to the given type is within bounds defined in the constraint. An important use of the schema in RDF is the validation whether those constraints are satisfied over the graph verifying that a node assigned to a given type satisfies the constraints defined in the type. Among the different schemas for RDF graph that have been defined we focus on two of them: *Shape Constraint Language* and *Shape Expression Schema*. Both schema languages are based on the notion of *shapes* that is analogy to DTD [Bray *et al.* 2008] and XML Schema (XSD) [Thompson *et al.* 2012]. SHACL syntax is RDF while ShEx syntax follows a more human-readable grammar defined in [Staworko *et al.* 2015].

Example 1.5.1. Consider the following shape constraints that capture the structure of the graph G_1 in Figure 1.17. We define four *shapes* ShTopic, ShAddress, ShUniversity and ShResearcher. We present the set of *shape definitions* using an abstract syntax as follows:

$$
\begin{aligned}
\mathsf{ShTopic} \rightarrow\ & \mathsf{ex{:}name} :: Lit^{1} \\
\mathsf{ShAddress} \rightarrow\ & \mathsf{ex{:}city} :: Lit^{1}; \\
& \mathsf{ex{:}zip} :: Lit^{1} \\
\mathsf{ShUniversity} \rightarrow\ & \mathsf{ex{:}name} :: Lit^{1}; \\
& \mathsf{ex{:}address} :: \mathsf{ShAddress}^{1} \\
\mathsf{ShResearcher} \rightarrow\ & \mathsf{foaf{:}name} :: Lit^{1}; \\
& \mathsf{foaf{:}knows} :: \mathsf{ShResearcher}^{*}; \\
& \mathsf{ex{:}masters} :: \mathsf{ShTopic}^{+}; \\
& \mathsf{ex{:}authors} :: Lit^{*}; \\
& \mathsf{ex{:}worksIn} :: \mathsf{ShUniversity}^{1}; \\
& \mathsf{foaf{:}mbox} :: Lit^{?}
\end{aligned}
$$

A shape definition describes a set of *triple constraints* that restricts the neighborhood with a specific *label*, the type of the target node and a *cardinality* that limits the number of outgoing edges with such label. We point out that the labels used in each definition are IRIs and the set of prefixes are defined in Table 1.1. Now, we describe how graph G_1 satisfies the shapes schema presented above by describing each shape definition and how G_1 satisfies it.

The shape ShTopic requires:

- exactly one edge labeled with ex:name to a literal node.

In G_1, the nodes dbp:Mathematics and dbp:Logic satisfy the shape constraint ShTopic because for each node there is only one outgoing edge labeled with ex:name.

The shape ShAddress requires:

- exactly one edge labeled with ex:city to a literal node; and
- exactly one edge labeled with ex:zip to a literal node.

In G_1, the node $\_{:}b_1$ satisfies the shape constraint ShAddress because there are only two outgoing edges with labels ex:city and ex:zip.

The shape ShUniversity requires:

- exactly one edge labeled with ex:name to a literal node; and
- exactly one edge labeled with ex:address to node of type ShAddress.

In G_1, the node ulille: satisfies ShUniversity because there is only one outgoing edge with label ex:name and one outgoing edge with label ex:address.

The shape ShResearcher requires:

- exactly one edge labeled with foaf:name to a literal node;
- zero or more edges labeled with foaf:knows to nodes that satisfy shape ShResearcher;
- one or more edges labeled with ex:masters to nodes that satisfy shape ShTopic;
- zero or more edges labeled with ex:authors to literal nodes;
- exactly one edge labeled with ex:worksIn to a node that satisfies shape ShUniversity; and
- zero or one edges labeled with foaf:mbox to a literal node.

The node ulille:jlozano satisfies the triple constraints of ShResearcher on the labels foaf:name, foaf:knows, ex:masters, ex:worksIn and foaf:mbox because the number of outgoing edges with these labels is within the bounds of the interval defined by each triple constraint. The constraint with label ex:authors also is satisfied even if there is no outgoing edge with this label since the minimum number of outgoing edges is zero. For the node ulille:etupac, it is easy to see that the triple constraints of ShResearcher on the labels foaf:name, ex:masters, ex:authors, ex:worksIn are satisfied. The triple constraints with labels foaf:knows and foaf:mbox are satisfied even if there are no outgoing edges because the minimum number of outgoing edges is zero.

We point out the fact that there is an outgoing edge with ex:age from ulille:jlozano does not make that this node not to satisfy ShResearcher since there is no constraint with label ex:age in this shape. Finally, the literal nodes of G_1 are assigned to a shape called Lit that does not constraint the neighbors of literal nodes. We say that the graph

```
ex:ShTopic
    a sh:NodeShape;
    sh:targetNode
        dbp:Mathematics,
        dbp:Logic;
    sh:Property[
        sh:path ex:name;
        sh:minCount 1;
        sh:maxCount 1;
        sh:datatype xsd:Lit].
ex:ShAddress
    a sh:NodeShape;
    sh:targetNode _:b1 ;
    sh:Property[
        sh:path ex:city;
        sh:minCount 1;
        sh:maxCount 1;
        sh:datatype xsd:Lit];
    sh:Property[
        sh:path ex:zip;
        sh:minCount 1;
        sh:maxCount 1;
        sh:datatype xsd:Lit].
ex:ShUniversity
    a sh:NodeShape;
    sh:targetNode ulille: ;
    sh:Property[
        sh:path ulille:name;
        sh:minCount 1;
        sh:maxCount 1;
        sh:datatype xsd:Lit];
    sh:Property[
        sh:path ex:address;
        sh:node ex:ShAddress;
        sh:maxCount 1;
        sh:datatype IRI].

ex:ShResearcher
    a sh:NodeShape;
    sh:targetNode
        ulille:jlozano,
        ulille:etupac;
    sh:Property[
        sh:path foaf:name;
        sh:minCount 1;
        sh:maxCount 1;
        sh:datatype xsd:Lit];
    sh:Property[
        sh:path ex:masters;
        sh:minCount 1;
        sh:node ex:ShTopic];
    sh:Property[
        sh:path foaf:knows;
        sh:node ex:ShResearcher];
    sh:Property[
        sh:path ex:authors;
        sh:datatype xsd:Lit];
    sh:Property[
        sh:path ex:worksIn;
        sh:minCount 1;
        sh:maxCount 1;
        sh:node ex:ShUniversity];
    sh:Property[
        sh:path foaf:mbox;
        sh:minCount 0;
        sh:maxCount 1;
        sh:datatype xsd:Lit].
```

Figure 1.21: Example of schema for RDF written in SHACL where prefixes are described in Table 1.1.

G_1 satisfies the shapes schema because for every shape and every node associated with a shape, the set of constraints is satisfied on the outgoing edges of the node. □

In the above exampe, if we define in XSD the type Lit then these shape definitions can be written in SHACL as shown in Figure 1.21 and ShEx as shown in Figure 1.22. As we see in these figures, the shape names are IRIs where ex is the prefix defined in Table 1.1 and the rest of an IRI is a name, but in the abstract formalism for ease of notation, we use only the name.

```
ex:ShTopic{
    ex:name xsd:Lit
}
ex:ShAddress{
    ex:city xsd:Lit;
    ex:zip xsd:Lit
}
ex:ShUniversity{
    ulille:name xsd:Lit;
    ex:address @ex:ShAddress
}
ex:Researcher{
    foaf:name xsd:Lit;
    foaf:knows @ex:Researcher *;
    foaf:mbox xsd:Lit ?;
    ex:authors xsd:Lit *;
    ex:worksIn @ex:University;
    ex:masters @ex:Topic +
}
```

Figure 1.22: Example of schema for RDF written in ShEx where prefixes are described in Table 1.1.

Now, we describe differences between SHACL and ShEx. We point out that in SHACL we have to declare which nodes are going to be validated in the shape. For instance the triple

$$(\mathsf{ex{:}ShTopic}, \mathsf{sh{:}targetNode}, \mathsf{dbp{:}Mathematics})$$

specifies that the node dbp:Mathematics must be validated with respect to the shape ShTopic. Also, a triple constraint in SHACL is called a *property shape*. Compared to SHACL, ShEx does not require the nodes to be specified in the definition while SHACL does. Also, ShEx does not use too many words to express a triple constraint. This makes an easy comprehension of the constraint language. Also, another difference is that ShEx defines a validation method that associates a shape to a node when the node satisfies the constraints imposed by the shape while SHACL validates a graph from the target nodes declared for a shape.

1.5.1 Typed RDF graph

Validation of graphs requires assigning type names to nodes, in particular, with a *typing* function. Together the graph and the typing function is called a *typed RDF graph*. We illustrate the notion of *typed RDF graph* with the following example.

Example 1.5.2. Consider the RDF graph G_0 in Figure 1.17 and the set of type names

$$\mathscr{T}_0 = \{\mathsf{ShResearcher}, \mathsf{ShTopic}, \mathsf{ShUniversity}, \mathsf{ShAddress}\}.$$

We define a $typing_0$ function for the nodes of G_0.

$$\begin{aligned} typing_0(\mathsf{dbp{:}Mathematics}) &= \{\mathsf{ShTopic}\} \\ typing_0(\_{:}\mathsf{b}_1) &= \{\mathsf{ShAddress}\} \\ typing_0(\mathsf{ulille{:}jlozano}) &= \{\mathsf{ShResearcher}\} \\ typing_0(\mathsf{ulille{:}}) &= \{\mathsf{ShUniversity}\} \\ typing_0(\text{“Jose Lozano”}) &= \{Lit\} \end{aligned}$$

Then the pair $(G_0, typing_0)$ is called a typed RDF graph. □

Let $\mathscr{T} \subseteq \mathscr{V}_R^{(1)}$ be a finite set of type names. A *$\mathscr{T}$-typed graph* is a pair $(G, typing)$, where G is a graph and $typing : nodes(G) \to 2^{(\mathscr{T} \cup \{Lit\})}$ is a function that assigns to every node of the graph G a (possibly empty) set of types, where Lit is the special type for literal nodes.

1.5.2 Shape constraints language

We now introduce the abstract Shape Constraint Language. Its syntax is very similar to ShEx compact syntax. We fix a finite set of shape names $\mathscr{T} \subseteq \mathscr{V}_R^{(1)}$. A *triple constraint* over $\mathscr{T} \cup \{Lit\}$ is a tuple $(p, T, \mu) \in \mathsf{Iri} \times (\mathscr{T} \cup \{Lit\}) \times \{\mathtt{1}, \mathtt{?}, \mathtt{*}, \mathtt{+}\}$, where

- p is referred to as a *property label*,
- T is referred to as a *target shape*

- μ is referred to as a *multiplicity* that constraints the number of occurrences of outgoing edges. We interpret multiplicities with intervals as follows (an interval $[n; m]$ is a finite representation of the set $\{k \mid n \leq k \leq m\}$).
  - `1` is exactly one and its corresponding interval is $[1; 1]$,
  - `?` is at most one and its corresponding interval is $[0; 1]$,
  - `*` is any and its corresponding interval is $[0; \infty]$, and
  - `+` is at least one and its corresponding interval is $[1; \infty]$.

  The intervals `1`, `?`, `*`, `+` are called *basic intervals*.

We often write a triple constraint (p, T, μ) as $p :: T^{\mu}$. We denote the set of triple constraints over a set of types $\mathscr{T} \cup \{Lit\}$ by $triConstr$.

In this manuscript (except for Chapter 6), shapes schemas are interpreted under open interpretation, which means that a graph satisfies a shapes schema even if there is a triple that is not constrained by the shapes schema. Now, a *shapes schema* is a pair $\mathbf{S} = (\mathscr{T}, \delta)$, where

- $\mathscr{T}$ is a finite set of shapes, and
- $\delta : \mathscr{T} \rightarrow \mathscr{P}(triConstr_{\mathscr{T} \cup \{Lit\}})$ is a function called *shape definition* that maps every symbol $T \in \mathscr{T}$ to a finite set of triple constraints over $\mathscr{T} \cup \{Lit\}$.

In this manuscript, we focus on deterministic shapes schemas, i.e. for every type name T and for every property label $p \in \mathsf{Iri}$ there is at most one triple constraint using p.

Now, we define the semantics of shapes schema. Given a graph G, the *outbound neighborhood* of a node $n \in nodes(G)$ with respect to an edge label $p \in \mathsf{Iri}$ is

$$out_G(n, p) = \{o \mid (n, p, o) \in G\}.$$

Take a $\mathscr{T}$-typed graph $(G, typing)$, a node n, and a triple constraint $p :: T^{\mu}$. The node n satisfies the triple constraint $p :: T^{\mu}$ if

- every neighbor node of n with edge label p has type T, i.e., for any $m \in out_G(n, p)$ it holds that $T \in typing(m)$; and

- the cardinality of $out_G(n, p)$ is within the interval of μ.

A $\mathscr{T}$-typed graph $(G, typing)$ *satisfies* a shapes schema $\mathbf{S} = (\mathscr{T}, \delta)$ if for any $n \in nodes(G)$ and any $T \in typing(n)$ the node n satisfies every triple constraint in $\delta(T)$. Also, we define the language of a shapes schema as the set $L(\mathbf{S}) = \{(G, typing) \mid (G, typing) \models \mathbf{S}\}$.

We define a graphical representation of shapes schema called *shape graph*, denoted by $G_{\mathbf{S}}$. A shape graph of a shapes schema $\mathbf{S} = (\mathscr{T}, \delta)$ over $\mathscr{T}$ is a pair $G_{\mathbf{S}} = (\mathscr{T} \cup \{Lit\}, E_{\mathbf{S}})$ where $\mathscr{T} \cup \{Lit\}$ is the set nodes and $E_{\mathbf{S}}$ is the set of edges with two labels defined as follows.

$$E_{\mathbf{S}} = \{(T, p, \mu, S) \in \mathscr{T} \times \mathsf{Iri} \times \{\mathtt{0}, \mathtt{1}, \mathtt{?}, \mathtt{*}, \mathtt{+}\} \times (\mathscr{T} \cup \{Lit\}) \mid p :: S^{\mu} \in \delta(T)\},$$

where nodes are T, S and (p, μ) is the pair of labels of the edge. For instance, the shape graph of the shapes schema of Example 1.5.1 is shown in Figure 1.23.

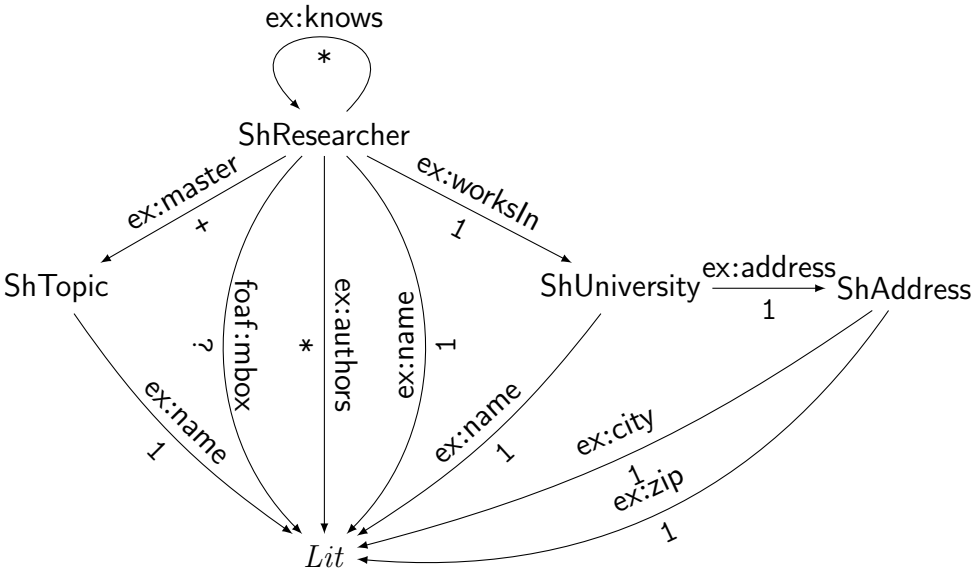


Figure 1.23: Shape graph of shapes schema in Example 1.5.1.

1.5.3 Logic formalization

The relational signature of a shapes schema $\mathbf{S}$ and a typing graph $(G, typing)$ over a set of types $\mathscr{T}$ is $\mathscr{R}_{\mathbf{S}} = \mathscr{T} \cup \{Lit\}$. We observe that any $\mathscr{T}$-typed graph $(G, typing)$ can be easily converted to a relational structure over the relational signature $\mathscr{G}_{\mathscr{T}} = \{Triple\} \cup$

$\mathscr{T} \cup \{Lit\}$. Consequently, in the sequel, we may view a $\mathscr{T}$-typed graph $(G, typing)$ as the corresponding relational structure over $\mathscr{G}_{\mathscr{T}}$. For a given shapes schema $\mathbf{S}$ and a typed graph $(G, typing)$, we define the relational structure $M_{\mathbf{S}}$ over $\mathscr{R}_{\mathbf{S}} \cup \mathscr{G}_{\mathscr{T}}$ corresponding to $(G, typing)$ with

$$T^{M_{\mathbf{S}}} = \{n \in nodes(G) \mid T \in typing(n)\} \text{ for any } T \in \mathscr{T},$$
$$Lit^{M_{\mathbf{S}}} = \{n \in nodes(G) \cap \mathsf{Lit} \mid Lit \in typing(n)\}.$$

Additionally, we define a function called $types$ over the relational structure of a typed RDF graph G where its input is a node and outputs the types associated in the graph, in symbols

$$types_G(n) = \{T \mid T(n) \in G\}.$$

1.5.4 Shape constraints as dependencies

A deterministic shapes schema $\mathbf{S}$ over a set of types $\mathscr{T}$ can be captured with a set $\Sigma_{\mathbf{S}}$ of dependencies. Next, we define the following rules that are used to capture a triple constraint $p :: S^{\mu}$ that is in the definition of a type T:

- the *type propagation* rule:

$$\mathbf{TP}(T, p, S) = \forall x, y.\ T(x) \wedge Triple(x, p, y) \Rightarrow S(y),$$

- the *predicate existence* rule:

$$\mathbf{PE}(T, p) = \forall x.\ T(x) \Rightarrow \exists y.\ Triple(x, p, y),$$

- the *predicate functionality* rule:

$$\mathbf{PF}(T, p) = \forall x, y, z.\ T(x) \wedge Triple(x, p, y) \wedge Triple(x, p, z) \Rightarrow y = z\ .$$

We point out that in terms of the classical relational data exchange, **TP** and **PE** are *target tuple generating dependencies* (*t-tgd*s), and **PF** is an *equality generating dependency*

(*egd*). We capture the shapes schema $\mathbf{S}$ with the following set of dependencies:

$$\begin{aligned}\Sigma_{\mathbf{S}} = \{&\mathbf{TP}(T, p, S) \mid T \in \mathscr{T},\ p :: S^{\mu} \in \delta(T)\} \cup \\ \{&\mathbf{PE}(T, p) \mid T \in \mathscr{T},\ p :: S^{\mu} \in \delta(T),\ \mu \in \{\mathtt{1}, \mathtt{+}\}\} \cup \\ \{&\mathbf{PF}(T, p) \mid T \in \mathscr{T},\ p :: S^{\mu} \in \delta(T),\ \mu \in \{\mathtt{1}, \mathtt{?}\}\}.\end{aligned}$$

In the sequel, the set of **TP**-dependencies of $\mathbf{S}$ is denoted by $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$, the set of **PE**-dependencies of $\mathbf{S}$ by $\Sigma_{\mathbf{S}}^{\mathbf{PE}}$ and the set of **PF**-dependencies of $\mathbf{S}$ by $\Sigma_{\mathbf{S}}^{\mathbf{PF}}$. We state the following lemma to prove that a given shapes schema is equivalent to a set of **TP**, **PF**, and **PE** dependencies.

Lemma 1.5.1. For every shapes schema $\mathbf{S} = (\mathscr{T}, \delta)$ and every $\mathscr{T}$-typed RDF graph $(G, typing)$, $(G, typing)$ satisfies $\mathbf{S}$ iff $(G, typing) \models \Sigma_{\mathbf{S}}$.

Proof. We fix a $\mathscr{T}$-typed graph $(G, typing)$ and shapes schema $\mathbf{S} = (\mathscr{T}, \delta)$.

For the $\Rightarrow$ direction, we prove by contraposition. Assume that $(G, typing) \not\models \Sigma_{\mathbf{S}}$. Our goal is to prove $(G, typing)$ does not satisfies $\mathbf{S}$. By definition of entailment, there is a dependency $\sigma \in \Sigma_{\mathbf{S}}$ that is not satisfied. The dependency σ can be of the following forms:

- $\mathbf{TP}(T_s, p, T_o)$. By construction of $\Sigma_{\mathbf{S}}$, the dependency σ occurs when a triple constraint is of the form $p :: T_o^{\mu}$ where $\mu \in \{\mathtt{1}, \mathtt{?}, \mathtt{*}, \mathtt{+}\}$. Since σ is not satisfied, $(s, p, o) \in G$ and $T_s \in typing(s)$. Because the node $o \in nodes(G)$, it must hold $T_o \in typing(o)$. But this fact is not, then the $\mathscr{T}$-typed graph $(G, typing)$ does not satisfy $\mathbf{S}$.

- $\mathbf{PE}(T_s, p)$. By construction of $\Sigma_{\mathbf{S}}$, the dependency σ occurs when a triple constraint is of the form $p :: T_o^{\mu}$ for some $\mu \in \{\mathtt{1}, \mathtt{+}\}$ and some $T_o \in \mathscr{T}$. Since σ is not satisfied, then $T_s \in typing(n)$ for some node $n \in nodes(G)$. Because the cardinality of outbound neighborhood of n w.r.t. an edge label p is 0, the $\mathscr{T}$-typed graph $(G, typing)$ does not satisfy $\mathbf{S}$.

- $\mathbf{PF}(T_s, p)$. By construction of $\Sigma_{\mathbf{S}}$, the dependency σ occurs when a triple constraint is of the form $p :: T_o^{\mu}$ for some $\mu \in \{\mathtt{1}, \mathtt{?}\}$ and some $T_o \in \mathscr{T}$. Since σ is

not satisfied, we have that $(s, p, o_1) \in G$ and $(s, p, o_2) \in G$ and $T_s \in \mathit{typing}(s)$, which makes the $\mathscr{T}$-typed graph (G, typing) to not satisfy $\mathbf{S}$.

For the $\Leftarrow$ direction, assume that $(G, \mathit{typing}) \models \Sigma_{\mathbf{S}}$. Our goal is to prove (G, typing) satisfies $\mathbf{S}$. We prove by contradiction. Suppose that (G, typing) does not satisfy $\mathbf{S}$. Then there is a node $n \in \mathit{nodes}(G)$ and $T \in \mathit{typing}(n)$ for which there is a triple constraint $p :: S^{\mu} \in \delta(T)$ that is not satisfied. This can occur because of the following two cases:

- There is a triple $(n, p, m) \in G$ such that $S \notin \mathit{typing}(m)$. Since $T(n)$ and $\mathit{Triple}(n, p, m)$ are facts in (G, typing) and $(G, \mathit{typing}) \models \mathbf{TP}(T, p, S)$, then $S(m)$ is a fact in (G, typing). Thus, $S \in \mathit{typing}(m)$; a contradiction.

- The cardinality of the outbound neighborhood of n w.r.t. p is not in the interval of μ.

  - When $\mu = \texttt{1}$ and $|\mathit{out}_G(n, p)| \neq 1$. It follows that $\mathbf{PF}(T, p) \in \Sigma_{\mathbf{S}}$ and $\mathbf{PE}(T, p) \in \Sigma_{\mathbf{S}}$. Since $(G, \mathit{typing}) \models \Sigma_{\mathbf{S}}$, then $|\mathit{out}_G(n, p)| = 1$; a contradiction.

  - When $\mu = \texttt{?}$ and $|\mathit{out}_G(n, p)| > 1$. It follows that $\mathbf{PF}(T, p) \in \Sigma_{\mathbf{S}}$. Since $(G, \mathit{typing}) \models \Sigma_{\mathbf{S}}$, then $|\mathit{out}_G(n, p)| \leq 1$; a contradiction.

  - When $\mu = \texttt{+}$ and $|\mathit{out}_G(n, p)| < 1$. It follows that $\mathbf{PE}(T, p) \in \Sigma_{\mathbf{S}}$. Since $(G, \mathit{typing}) \models \Sigma_{\mathbf{S}}$, then $|\mathit{out}_G(n, p)| \geq 1$; a contradiction.

□

Chapter 2

Relational to RDF data exchange

In this chapter, we present the framework for relational to RDF data exchange and the problems that we address in this manuscript.

2.1 Relational to RDF data exchange setting

In this section, we formalize the relational database to RDF graph data exchange problem and, compare to relational data exchange, we identify two new challenges: converting data values from databases to IRIs and associating the shapes schema for constraining the RDF graph. For the first challenge, we use *IRI constructors*. An IRI constructor is a function that takes a data value and converts it to a IRI. For the second challenge, we use the signature of shapes schema in the rule specification. We illustrate our approach with the following example.

Example 2.1.1. Recall the database schema of the academic institute in Example 1.2.1 which has six relations:

UniTeam($\underline{Team, University}, Place$),

Researcher($\underline{Idr}, Name, Email, Expertise, Team$),

Talk($\underline{Idr, Idc, Title}$)

Conference($\underline{Idc}, Name, Year, Place$),

Registration($\underline{Idr, Idc}, Date$),

Collaborator($\underline{Idr, Idc, Title, Idrc}$).

and the set of inclusion dependencies:

$$\textit{Registration}[Idc] \subseteq \textit{Conference}[Idc]$$
$$\textit{Registration}[Idp] \subseteq \textit{Researcher}[Idr]$$
$$\textit{Talk}[Idp\ Idc] \subseteq \textit{Registration}[Idr\ Idc]$$
$$\textit{Collaborator}[Idr\ Idc\ Title] \subseteq \textit{Talk}[Idr\ Idc\ Title]$$
$$\textit{Researcher}[Team] \subseteq \textit{UniTeam}[Team]$$

$$\begin{aligned}
\mathsf{ShUniversity} \rightarrow\ & \mathsf{ex{:}name} :: Lit^{1};\\
& \mathsf{ex{:}address} :: \mathsf{ShAddress}^{1}\\
\mathsf{ShAddress} \rightarrow\ & \mathsf{ex{:}city} :: Lit^{1};\\
& \mathsf{ex{:}zip} :: Lit^{1}\\
\mathsf{ShTopic} \rightarrow\ & \mathsf{ex{:}name} :: Lit^{1}\\
\mathsf{ShResearcher} \rightarrow\ & \mathsf{foaf{:}name} :: Lit^{1};\\
& \mathsf{ex{:}masters} :: \mathsf{ShTopic}^{+};\\
& \mathsf{ex{:}worksIn} :: \mathsf{ShUniversity}^{1};\\
& \mathsf{ex{:}authors} :: \mathsf{ShPaper}^{+};\\
& \mathsf{foaf{:}knows} :: \mathsf{ShResearcher}^{*};\\
& \mathsf{foaf{:}mbox} :: Lit^{?}\\
\mathsf{ShPaper} \rightarrow\ & \mathsf{ex{:}name} :: Lit^{1};\\
& \mathsf{ex{:}in} :: \mathsf{ShConference}^{1}\\
\mathsf{ShConference} \rightarrow\ & \mathsf{ex{:}name} :: Lit^{1};\\
& \mathsf{ex{:}year} :: Lit^{1};\\
& \mathsf{ex{:}place} :: Lit^{1};\\
& \mathsf{ex{:}chair} :: \mathsf{ShResearcher}^{+};\\
& \mathsf{ex{:}sponsor} :: \mathsf{ShUniversity}^{*}
\end{aligned}$$

Figure 2.1: Example of shapes schema.

We show how to define mappings between a database schema and a shapes schema. We want to export this database to an RDF, where we want to export universities, researchers, papers and conferences. For each university, we want to export its researchers and for each researcher their papers. Also we want to map the experience of a researcher on a topic, we want to export the information on conferences where the paper was published and the fact that a researcher knows other researcher when they wrote the same

paper. The graph must follow the structure that is described in Figure 2.1.

We recall that RDF requires IRIs as their nodes and the information stored in the database are data values. IRI constructors solve the problem of mapping data values to IRIs. We can define a range of IRI constructors according to the needs. The IRI constructors are used for the task of specifying mappings from a relational schema to a shapes schema.

Example 2.1.2 (cont. Example 2.1.1)**.** We define the following IRI constructors and we point out that each constructor is associated to a shape name:

- $f_{uni2iri}$ takes the name of the university and replaces blank spaces with character "–". Then the function adds with the prefix univ:. For instance,

$$f_{uni2iri}(\textit{University of Lille}) = \mathsf{univ{:}university{-}of{-}lille};$$

- $f_{top2iri}$ adds the prefix dbp: with the name of the topic. For instance,

$$f_{top2iri}(\textit{Maths}) = \mathsf{dbp{:}maths};$$

- $f_{res2iri}$ takes the id of the researcher and retrieves from the academic institute database the name associated to the id. Then, this function generates a shorten identifier for this name. Finally, the function adds the prefix ulille:res/. For instance for *id* with value 1, the name is *jose lozano* and the function returning:

$$f_{res2iri}(1) = \mathsf{ulille{:}res/jlozano};$$

- $f_{pap2iri}$ takes the name and year of a *Conference* and the title of a *Talk* presented in the given conference. It outputs an IRI consisting of the prefix dblp:paper# with a unique integer associated to the paper. For instance,

$$f_{pap2iri}(\textit{icdt}, 2019, \textit{Tutoring Web math platform}) = \mathsf{dblp{:}paper\#1};$$
$$f_{pap2iri}(\textit{icdt}, 2019, \textit{Skolemization of prenex formulas}) = \mathsf{dblp{:}paper\#2};$$
$$f_{pap2iri}(\textit{iswc}, 2019, \textit{On algebraic connectivity of graphs}) = \mathsf{dblp{:}paper\#3}.$$

- $f_{con2iri}$ takes the name and year of the conference and adds the prefix dblp:. For instance,

$$f_{con2iri}(\textit{ICDT}, 2019) = \mathsf{dblp{:}icdt2019}. \quad \square$$

Now, we introduce nodes with IRI constructors in the mapping definition. We also need to associate types to the nodes in the transformation rules.

Example 2.1.3 (cont. Example 2.1.2)**.** In this case, the set of transformation rules are specified as follows:

- a university is mapped to a node of shape ShUniversity. Formally,

$$\textit{UniTeam}(x_1, x_2, x_3) \Rightarrow \mathit{Triple}(f_{uni2iri}(x_2), \mathsf{ex{:}name}, x_2) \wedge \mathsf{ShUniversity}(f_{uni2iri}(x_2)); \quad (2.1)$$

- the expertise of researcher is mapped to a node of shape ShTopic. Formally,

$$\textit{Researcher}(x_1, x_2, x_3, x_4, x_5) \Rightarrow \mathit{Triple}(f_{top2iri}(x_4), \mathsf{ex{:}name}, x_4) \wedge \mathsf{ShTopic}(f_{top2iri}(x_4)); \quad (2.2)$$

- a researcher is mapped to a node of shape ShResearcher with the outgoing edges that identify name (foaf:name), mail (foaf:mbox), and a topic where the researcher is an expert (ex:masters). Formally,

$$\begin{aligned}\textit{Researcher}(x_1, x_2, x_3, x_4, x_5) \Rightarrow\ & \mathit{Triple}(f_{res2iri}(x_1), \mathsf{foaf{:}name}, x_2) \wedge \\ & \mathit{Triple}(f_{res2iri}(x_1), \mathsf{ex{:}masters}, f_{top2iri}(x_4)) \wedge \\ & \mathit{Triple}(f_{res2iri}(x_1), \mathsf{foaf{:}mbox}, x_3) \wedge \\ & \mathsf{ShResearcher}(f_{res2iri}(x_1)); \end{aligned} \quad (2.3)$$

- a talk is mapped to a node of shape ShPaper with an outgoing edge ex:in that identify the name of the paper (ex:name) and the conference where talk is presented

(ex:in). Formally,

$$\begin{aligned}\textit{Talk}(x_1, x_2, x_3) \wedge \textit{Conference}(x_2, x_4, x_5, x_6) \Rightarrow \\ \textit{Triple}(f_{pap2iri}(x_4, x_5, x_3), \mathsf{ex{:}name}, x_3) \wedge \\ \textit{Triple}(f_{pap2iri}(x_4, x_5, x_3), \mathsf{ex{:}in}, f_{con2iri}(x_4, x_5)) \wedge \\ \mathsf{ShPaper}(f_{pap2iri}(x_4, x_5, x_3)) \end{aligned} \tag{2.4}$$

- a conference is mapped to a node of shape ShConference with its name, year and place. Formally,

$$\begin{aligned}\textit{Conference}(x_1, x_2, x_3, x_4) \Rightarrow \textit{Triple}(f_{con2iri}(x_2, x_3), \mathsf{ex{:}name}, x_2) \wedge \\ \textit{Triple}(f_{con2iri}(x_2, x_3), \mathsf{ex{:}year}, x_3) \wedge \\ \textit{Triple}(f_{con2iri}(x_2, x_3), \mathsf{ex{:}place}, x_4) \wedge \\ \mathsf{ShConference}(f_{con2iri}(x_2, x_3)); \end{aligned} \tag{2.5}$$

- whenever a researcher is part of a team that corresponds to a university, there is an edge ex:worksIn between a node corresponding to the researcher and a node corresponding to the university. Formally,

$$\begin{aligned}\textit{Researcher}(x_1, x_2, x_3, x_4, x_5) \wedge \textit{UniTeam}(x_5, x_6, x_7) \Rightarrow \\ \textit{Triple}(f_{res2iri}(x_1), \mathsf{ex{:}worksIn}, f_{uni2iri}(x_6)); \end{aligned} \tag{2.6}$$

- for any talk presented or collaborated on by a researcher there is an edge from node corresponding to this researcher to the node corresponding to the talk. Formally, this involves two st-tgds respectively

$$\begin{aligned}\textit{Researcher}(x_1, x_2, x_3, x_4, x_5) \wedge \textit{Talk}(x_1, x_6, x_7) \wedge \textit{Conference}(x_6, x_8, x_9, x_{10}) \Rightarrow \\ \textit{Triple}(f_{res2iri}(x_1), \mathsf{ex{:}authors}, f_{pap2iri}(x_8, x_9, x_7)) \end{aligned} \tag{2.7}$$

$$\textit{Researcher}(x_1, x_2, x_3, x_4, x_5) \wedge$$
$$\textit{Collaborator}(x_6, x_7, x_8, x_1) \wedge \textit{Conference}(x_7, x_9, x_{10}, x_{11})$$
$$\Rightarrow \textit{Triple}(f_{res2iri}(x_1), \mathsf{ex{:}authors}, f_{pap2iri}(x_9, x_{10}, x_8)); \quad (2.8)$$

- and whenever a researcher and an other researcher authored a talk, there is an edge foaf:knows from the node corresponding to the former researcher to the node corresponding the later researcher. Formally,

$$\textit{Collaborator}(x_1, x_2, x_3, x_4) \Rightarrow \textit{Triple}(f_{res2iri}(x_1), \mathsf{foaf{:}knows}, f_{res2iri}(x_4)). \quad (2.9)$$

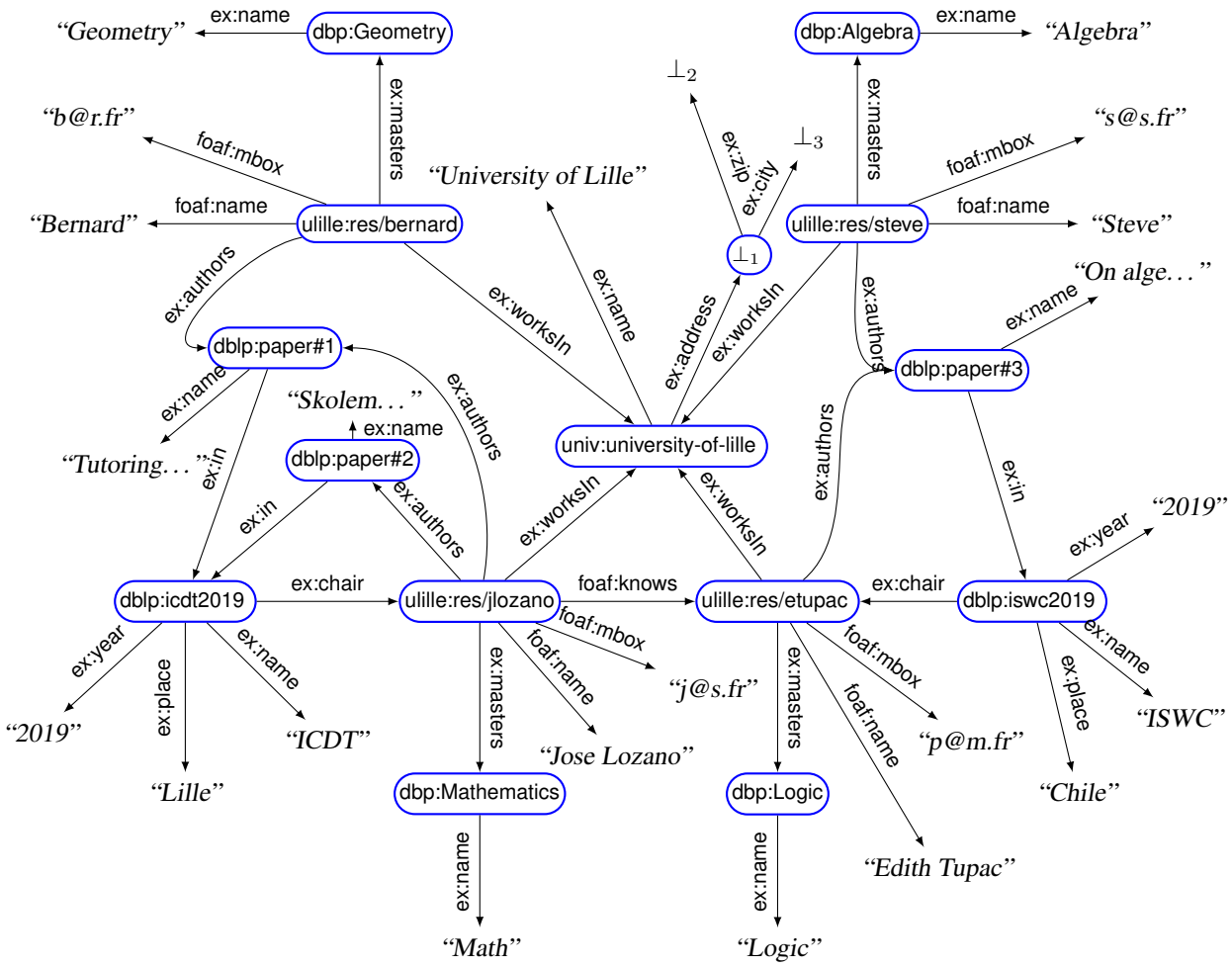


Figure 2.2: A solution for the Example 2.1.1.

The result of applying the rules is a typed graph composed of a graph and a typing.

In this data exchange setting, the graph is shown in Figure 2.2 and the typing is presented in Figure 2.3. □

ShTopic	ShUniversity
dbp:Mathematics dbp:Logic dbp:Algebra dbp:Geometry	univ:university−of−lille

ShResearcher	ShConference	ShPaper
ulille:res/jlozano ulille:res/etupac ulille:res/steve ulille:res/bernard	dblp:icdt2019 dblp:iswc2019	dblp:paper#1 dblp:paper#2 dblp:paper#3

Figure 2.3: Typing of graph in Example 2.2.

Formalization. We now formalize the data exchange setting that we work throughout this manuscript. We define a library of *IRI constructors* as a pair $\mathbf{F} = (\mathscr{F}, F)$, where $\mathscr{F}$ is a set of function names, and F is their interpretation i.e., a mapping that assigns to every IRI constructor name $f \in \mathscr{F}$ a function $f^F : \mathsf{ConstLit}^k \Rightarrow \mathsf{Iri}$, where k is the arity of f. A library $\mathbf{F}$ is *non-overlapping* if F assigns to elements of $\mathscr{F}$ injective functions with pairwise disjoint ranges. In the sequel, we consider settings where the library is non-overlapping.

We adapt the definition of relational data exchange to treat graphs and define the setting as follows.

Definition 2.1.4. A constructive *relational to RDF data exchange setting* is a tuple $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\mathrm{st}}, \mathbf{F})$, where

- $\mathbf{R} = (\mathscr{R}, \mathit{attrs}, \Sigma_{\mathrm{fd}}, \Sigma_{\mathrm{ind}})$ is a source relational schema,
- $\mathbf{S} = (\mathscr{T}, \delta)$ is a target shapes schema,
- $\mathbf{F} = (\mathscr{F}, F)$ is a library of non-overlapping *IRI constructors*,
- Σ_{st} is a set of *source-to-target tuple generating dependencies* (st-tgds) whose bodies are conjunction of atomic formulas over $\mathscr{R}$ and whose heads are conjunction of atomic formulas over $\mathscr{G}_{\mathscr{T}} \cup \mathscr{F}$ where $\mathscr{G}_{\mathscr{T}} = \{\mathit{Triple}, \mathit{Lit}\} \cup \mathscr{T}$ and these formulas do not contain existential variables and use IRI constructors.

Moreover, for any st-tgd $\sigma \in \Sigma_{\text{st}}$, we assume there is no monadic atom of the following form:

- $T(\mathbf{x})$ with sequence of variables of length one $\mathbf{x} \subseteq vars(\sigma)$ and $T \in \mathscr{T}$ in the head of σ and
- $Lit(f(\mathbf{x}))$ with some IRI constructor $f \in \mathscr{F}$ and some sequence of variables $\mathbf{x} \subseteq vars(\sigma)$. □

In the sequel, we adapt the definition of solution of relational data exchange as follows.

Definition 2.1.5 (Solution)**.** Take a data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$, and let I be a consistent instance of $\mathbf{R}$. Then, a *solution* to I w.r.t. $\mathscr{E}$ is any $\mathscr{T}$-typed graph J such that $I \cup J \cup F \models \Sigma_{\text{st}}$ and J satisfies the shapes schema $\mathbf{S}$. The set of solutions to I w.r.t. $\mathscr{E}$ is denoted as $sol_{\mathscr{E}}(I)$. □

Additionally, we consider the notion of *core pre-solution*, which is the result of exporting the relational data to triples with Σ_{st} and then propagating types with $\Sigma_{\mathbf{S}}^{\mathsf{TP}}$ (without creating any new nodes). We define the core pre-solution as follows.

Definition 2.1.6 (Core pre-solution)**.** The core pre-solution for I to $\mathscr{E}$ is the unique minimal graph J_0 that satisfies $J_0 \cup I \models \Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathsf{TP}}$, where $\Sigma_{\mathbf{S}}^{\mathsf{TP}}$ is the set of TP-dependencies generated by the shapes schema $\mathbf{S}$.

We observe that given a consistent instance I of $\mathbf{R}$ and constructive data exchange setting $\mathscr{E}$, the core pre-solution J_0 to I w.r.t. $\mathscr{E}$ always exists because there is always a finite chase sequence on I with Σ_{st} and $\Sigma_{\mathbf{S}}^{\mathsf{TP}}$. We have a finite chase sequence because the chase does not introduce any null values and there is no equality dependencies. By property of the chase with these kind of dependencies [Fagin *et al.* 2005a], the result is unique and minimal because no new null values are added.

2.2 R2RML: proof of concept

In this section, we describe the core definitions and assumptions of R2RML that provide rationale for using constructive data exchange to model the process of exporting data from relational databases to RDF. Following the definition of Rodríguez-Muro and

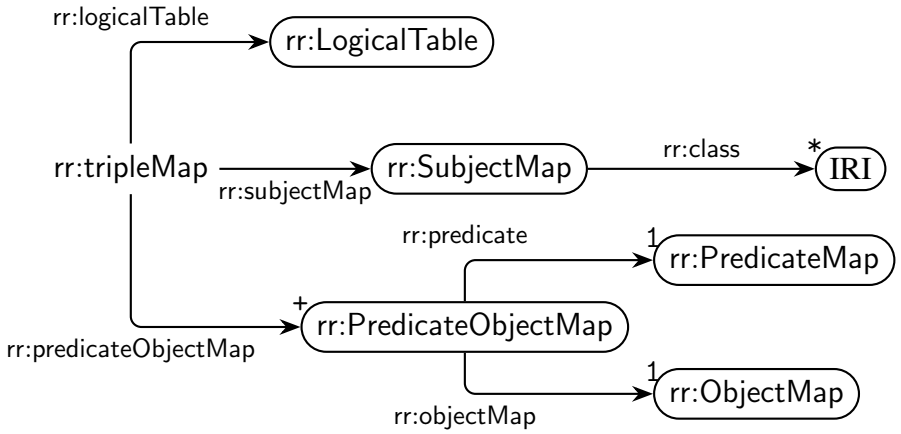


Figure 2.4: A well-formed R2RML mapping node from [Rodriguez-Muro & Rezk 2015].

Rezk [Rodriguez-Muro & Rezk 2015], a R2RML mapping is expressed as an RDF graph. The graph is not arbitrary, a well-formed mapping consists of one or more trees called *triple maps* with structure as shown in Fig 2.4. Each tree has a root node, called triple map node, which is connected to exactly one logical table node (rr:logicalTable), one subject map node (rr:subjectMap) and one or more predicate-object map nodes (rr:predicateObjectMap).

The logical table of a triple map is a SQL query and its result is mapped to RDF triples. It may be either (i) a table, (ii) or a complex SQL query. A table is represented by a resource that has exactly one rr:tableName property with the string denoting the table. A SQL query is represented by a resource with exactly one rr:sqlQuery property whose value is a SQL query string. A logical table row is a tuple obtained from the execution of the SQL query in the input database. Each row in the logical table entails a triple for every predicate object. Given a row, all triples generated from it share the same subject. Predicate, object and subject maps are constructed using term maps.

A term map specifies what is the RDF term used for a subject, predicate, or object. For a subject, the RDF term is always an IRI template, for the predicate the RDF term is an IRI value, for an object the RDF term is either an IRI template or RDF constant (literal or IRI value). An IRI template indicates how to construct an IRI using an IRI and attribute values from rows. These elements are represented as follows:

- If the element is an IRI template then it is represented by rr:template property whose value is an IRI that includes the name of attributes that come from the

SQL query or table result. An IRI template generates a unique IRI.

- If the element is an attribute name then it is represented by rr:column whose value is an attribute name.
- If the element is an RDF constant then it is represented by rr:constant whose value is an IRI or literal value.

A subject map may specify zero or more class IRI's represented by the rr:class property. The value of the property must be a valid IRI. A term map when is used in an object map optionally allows to specify the datatype (rr:datatype) in case of an attribute name and RDF constant. A predicate map is specified with rr:predicate property. Finally, the object map is specified with rr:objectMap.

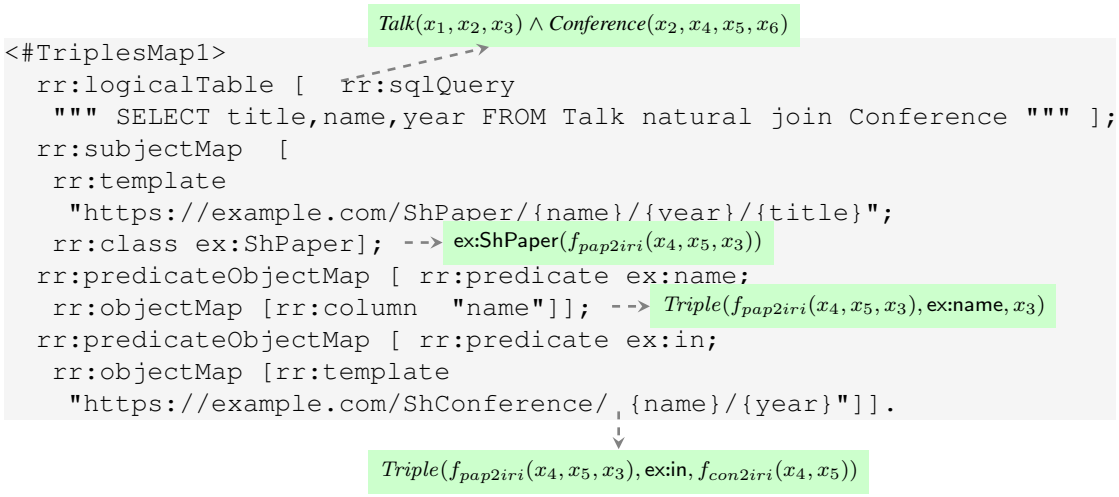


Figure 2.5: A R2RML mapping captured by a constructive st-tgd.

Now, we argue that constructive st-tgds are suitable for modeling a large fragment of R2RML. We show a translation of R2RML to constructive st-tgds.

$$\begin{aligned} &Talk(x_1, x_2, x_3) \wedge \\ &\quad Conference(x_2, x_4, x_5, x_6) \Rightarrow Triple(f_{pap2iri}(x_4, x_5, x_3), \mathsf{ex{:}name}, x_3) \wedge \\ &\qquad\qquad Triple(f_{pap2iri}(x_4, x_5, x_3), \mathsf{ex{:}in}, f_{con2iri}(x_4, x_5)) \wedge \\ &\qquad\qquad \mathsf{ex{:}ShPaper}(f_{pap2iri}(x_4, x_5, x_3)) \end{aligned}$$

Every triple map gives one constructive st-tgd. For instance, we consider the R2RML mapping shown in Figure 2.5 that exports contents of relational database presented in Example 1.1. This mapping takes the result of executing the query specified in

rr:sqlQuery and generates triples with subject nodes of class ex:ShPaper and two outgoing edges with labels ex:name and ex:in. The construction of a constructive st-tgd is as follows. We take the SQL query of rr:sqlQuery and convert to a conjunction of relational atoms where the shared variables are in the positions corresponding to the attributes used in the join or cross product with equality of attributes and the other positions are filled with fresh variables. If no join or cross product with equality of attributes is present in the query, then there are no shared variables. We only support natural, inner joins and cross product with equality of attributes. As a result, we have the following conjunction of atoms for the body of the constructive st-tgd

$$\textit{Talk}(x_1, x_2, x_3) \wedge \textit{Conference}(x_2, x_4, x_5, x_6)$$

where x_2 corresponds to the position of the attribute Idc used to do the natural join. We observe that the interpretation of this body of a st-tgd is different from the interpretation of the SQL query. However, if no attributes are specified in the projection, but it is used the symbol $*$, then the interpretation of the body of a st-tgd is the same as the interpretation of the query. We point out that SQL projections, where not all attributes are specified, are not captured by the body of a st-tgd, but the interpretation of a R2RML mapping generates the same result that the interpretation of a st-tgd generates.

Now, we take the subject map and assume it is represented by an IRI template. We create a function term with a function symbol where the arguments are the variables that appeared in the position of the attributes $name, year$ and $title$. In our mapping, we construct the following function term

$$f_{pap2iri}(x_4, x_5, x_3)$$

where x_4, x_5 corresponds to the attributes $name$ and $year$ of *Conference*, and x_3 corresponds to the attribute $title$ of *Talk*. Then, for each predicate object map, we create a triple atom where a function term is created for the subject map. We dedicate a unique function symbol to each IRI template and assign the IRI template as the interpretation of the function symbol. This function term is placed in the subject position of the triple atom. The value of the term map of the predicate map is placed in the predicate position

of the triple atom. Finally, we take the object map. If the term map is an IRI template, we create a function term as done for the subject map. If the term map is a value, then we place the value directly in the st-tgd. If the value is an attribute name, then we put the variable that appears in the position of the attribute of some relation being in the query. If the rr:class property is specified, then we create a monadic relational atom where the name of this relational atom is the name specified with rr:class and the argument of this monadic relational atom is the function term created for the subject map. Following this process, we obtain that the head of the constructive st-tgd is as follows:

$$\mathsf{ex{:}ShPaper}(f_{pap2iri}(x_4, x_5, x_3)) \wedge \mathit{Triple}(f_{pap2iri}(x_4, x_5, x_3), \mathsf{ex{:}name}, x_3) \wedge \mathit{Triple}(f_{pap2iri}(x_4, x_5, x_3), \mathsf{ex{:}in}, f_{con2iri}(x_4, x_5)).$$

We make several assumptions in the process above and not all R2RML follow those assumptions. Still we cover a large fragment of R2RML where no aggregate functions are present and no RDF data types are declared in the R2RML mappings. We discuss the limitations of our framework to capture R2RML: not every R2RML mapping has an equivalent constructive st-tgd interpretation. Concerning the SQL query, aggregate queries are not convertible to constructive st-tgds. For the object map, we do not capture the optional property that specifies the type of literal values because we do not have in the shapes schema. However, we can extend the framework by adding types to relational database and types to shapes schema but will require to lead with all the interoperability issues because XSD datatypes are not the same as SQL datatypes. Also this extension will arise technical complications in the study of consistency problem, but the complexity of the theoretical model of constructive st-tgds is the same. Therefore, for simplicity we abstract RDF datatypes for our consideration. Aggregation queries can not be handled in any extension of our framework.

2.3 Problems of interest

In this manuscript, we study several problems related to relational to RDF data exchange described below.

2.3.1 Checking consistency

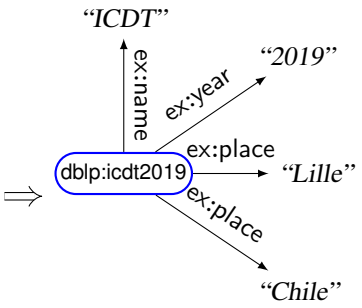
The ability to issue warnings that a data exchange setting does not admit solutions for some source instance would be the most welcome feature for a data administrator. A relational to RDF data exchange setting is consistent if for every source instance, there is a graph that satisfies the shapes schema.

Example 2.3.1. Consider the data exchange setting of Example 2.1.1. There is no solution for this instance in Figure 2.6a w.r.t. the data exchange setting. Therefore, this setting is not consistent. The problem springs from two tuples with *ICDT* name

Conference

Idc	*Name*	*Year*	*Place*
1	*ICDT*	2019	*Lille*
2	*ICDT*	2019	*Chile*

(a) Source instance.



(b) Target instance not satisfying schema.

Figure 2.6: Example of consistency problem.

and different places in the conference table. These tuples make the graph in Figure 2.6b contain two outgoing edges with the same label from node dblp:icdt. However, our shapes schema requires every conference to have precisely one place. □

We study techniques to identify if a relational to RDF data exchange setting is consistent in Chapter 3.

2.3.2 Computing certain answers

Given a relational to RDF data exchange setting, certain answers to query Q w.r.t. a source instance I are answers present in every solution for I and the data exchange setting. We distinguish two types of shapes schemas: non-recursive and recursive. Relational data exchange framework can be used to compute certain answers for non-recursive shapes schema. Indeed, non-recursive shapes schema can be expressed with

super-weakly acyclic tgds, which are known to have chase termination. This guarantees the existence of universal solution, which can be used to compute certain answers for a given query. On the other hand, recursive shapes schema are not equivalent to super-weakly acyclic tgds. Furthermore, a universal solution needs not to exist.

Example 2.3.2 (cont. Example 2.1.1)**.** The shapes schema of this Example 2.1.1 is recursive and, in fact, a finite universal solution for this source database instance I and data exchange setting $\mathscr{E}_1$ does not exist. The chase procedure for this setting will not terminate because node dblp:2019 requires to have a chair, which is not provided by the database. Thus a fresh node is created and associated to a type ShResearcher, which requires to be an author of at least one paper. Again, a fresh node is created and associated to a type ShPaper, which needs to be in a conference. Thus, a fresh node is created and associated to a type ShConference, and so on resulting in an infinite chase sequence as seen in Figure 2.7. □

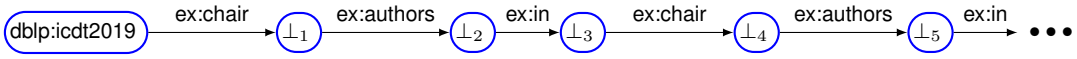


Figure 2.7: Part of the infinite solution for the database conference I w.r.t. $\mathscr{E}_1$.

In Chapter 4, we propose an alternative notion of universal simulation solution that is suitable for computing certain answers to a subclass of nested regular expressions [Pérez *et al.* 2010]. We study the complexity of computing certain answers and investigate a minimal-size universal simulation solution whose construction is in low complexity.

2.3.3 Visual mapping language

Non-expert users may have difficulty to learn mapping languages such the one presented in our framework. We propose a *visual mapping language* (VML) and an editor that allows to define VML mappings without using formal mapping language. We point out that VML captures a rich and expressive subset of our *constructive data exchange setting* proposed in Definition 2.1.4.

The specification starts with the tool visualizing in the left side the relational schema and in the right side the shapes schema and the users draw arrows that map objects and attributes from one schema to another as needed. The arrow colors correspond to its role, which are explored in detail in Section 5.4.

Example 2.3.3. Consider a source schema $\mathbf{R}_1$ with three relations *Product*($\underline{pid}, name$), *Supplier*($\underline{sid}, name$), and *ProdSup*($\underline{pid}, \underline{sid}$); and a target shapes schema defined as follows.

$$\mathsf{ShProd} \rightarrow \mathsf{ex{:}descr} :: Lit^1; \mathsf{ex{:}supplied} :: \mathsf{ShSupp}^+$$

$$\mathsf{ShSupp} \rightarrow \mathsf{ex{:}name} :: Lit^1$$

Figure 2.8 shows mappings from $\mathbf{R}_1$ to the shapes schemas described above. We study the accessibility of the proposed visual mapping language by a user evaluation. □

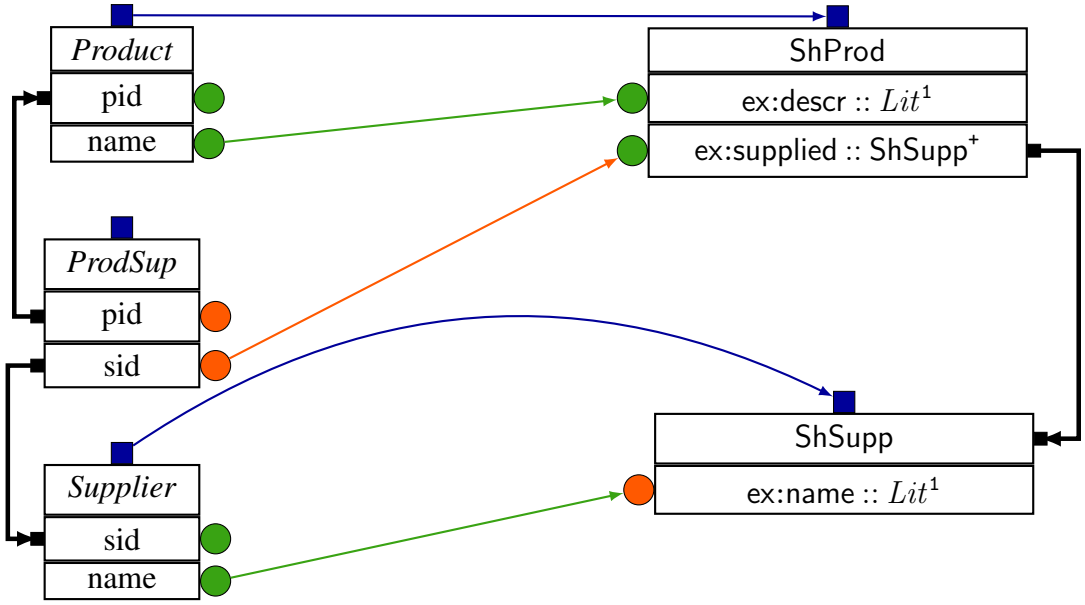


Figure 2.8: Graphical Mapping Language.

In Chapter 5, we describe ShERML, a tool that facilitates the specification of mappings with the use of a visual mapping language.

2.3.4 Schema elicitation

Relational to RDF data exchange setting is defined with the relational schema, set of mappings, and shapes schema. Since RDF is schema-less and R2RML has been introduced recently, then it is conceivable that many mappings have been written without schema. For a legacy reason, numerous settings have no target shapes schema provided. It is useful to have the shapes schema because it permits to understand the structure of the graph and to check if mappings are well-defined to obtain the desired graph.

Example 2.3.4. We are provided with a relational schema that has the following relations $\textit{Book}(\underline{isbn}, title, auth\_id)$, $\textit{Author}(\underline{auth\_id}, name)$ and $\textit{Bio}(\underline{auth\_id}, bio)$. The schema also has the following dependencies.

$$\textit{Bio}[auth\_id] \subseteq \textit{Author}[auth\_id]$$
$$\textit{Book}[auth\_id] \subseteq \textit{Author}[auth\_id]$$

Also, we are given a set of mappings.

$$\begin{aligned}
\textit{Book}(x_1, x_2, x_3) \Rightarrow\ & Triple(f_{book2iri}(x_1), \mathsf{ex{:}isbn}, x_1) \wedge \\
& Triple(f_{book2iri}(x_1), \mathsf{ex{:}title}, x_2) \wedge \\
& Triple(f_{book2iri}(x_1), \mathsf{ex{:}author}, f_{aut2iri}(x_3)); \\
\textit{Author}(x_1, x_2) \Rightarrow\ & Triple(f_{aut2iri}(x_1), \mathsf{ex{:}name}, x_2); \\
\textit{Author}(x_1, x_2) \wedge \textit{Book}(x_3, x_4, x_1) \Rightarrow\ & Triple(f_{aut2iri}(x_1), \mathsf{ex{:}book}, f_{book2iri}(x_3)); \\
\textit{Author}(x_1, x_2) \wedge \textit{Bio}(x_1, x_3) \Rightarrow\ & Triple(f_{aut2iri}(x_1), \mathsf{ex{:}bio}, x_3).
\end{aligned}$$

The following schema characterizes possible output RDF graphs produced by the mappings.

$$\mathsf{ShBook} \rightarrow \mathsf{ex{:}isbn} :: Lit^{\mathsf{1}}; \mathsf{ex{:}title} :: Lit^{\mathsf{1}}; \mathsf{ex{:}author} :: \mathsf{ShAuth}^{\mathsf{1}}$$
$$\mathsf{ShAuth} \rightarrow \mathsf{ex{:}name} :: Lit^{\mathsf{1}}; \mathsf{ex{:}bio} :: Lit^{\mathsf{?}}; \mathsf{ex{:}book} :: \mathsf{ShBook}^{*}$$

Indeed, this schema describes that for a book there might be precisely one isbn, title and it might be written by precisely one author who might have precisely one name, might have or not a biographical information and might have a collection of books potentially empty. This follows from mappings and relational schema. □

In Chapter 6, we investigate the problem of generating shapes schemas and identify two desirable properties for the output target schema. We propose an elicitation algorithm.

2.4 Related work

Here, we only describe related works on formalizing data exchange applications as data exchange frameworks. For every problem of interest, we will present related work independently in the corresponding sections.

Data exchange problem is studied in different contexts such as graph to graph [Barceló *et al.* 2013] or relational to graph [Boneva *et al.* 2015]. The frameworks proposed in each context varies according to the type of data that is treated. For our context of relational to RDF, none of the existent frameworks addresses relational to RDF with shapes schemas. We present a list of frameworks and the relation of each framework with our proposal.

The classical relational to relational data exchange framework, which is proposed by Fagin et al. [Fagin *et al.* 2005a], is composed of a relational source and target schema, source-to-target tuple generating dependencies and target dependencies. The authors define what is a solution for data exchange. We adapt their setting to our context and define a solution as Fagin et al. did.

XML data exchange [Arenas & Libkin 2008], proposed by Arenas et al., deals with hierarchical data. Their setting uses source and target DTDs and for the mappings a closer formalism to tree patterns [Amer-Yahia *et al.* 2002] and XPath [Benedikt *et al.* 2005]. Solutions are defined as in the relational case. Since DTDs define the structure of trees, we cannot use DTDs as a schema for graphs. Another approach of exchanging hierarchical data is with the use of nested dependencies [Fuxman *et al.* 2006] as mappings proposed by Fuxman et al. We took from nested dependencies the idea of using function symbols, but not Skolem functions because we fix the functions that are used in the constructive st-tgds.

Direct mapping (DM) [Arenas *et al.* 2012], proposed by W3C, is also a subclass of relational to relational data exchange that exchanges relational data to RDF. DM defines simple transformation rules to generate RDF from relational data such as a rule to generate the subject IRI or a rule to generate triples. The data exchange setting is a tuple composed of the relational source schema that contains the relational signature, primary keys and foreign keys, and a set of st-tgds where the head of each st-tgd is over the triple signature. Authors define properties that the set of st-tgds must satisfied

such that the graph contains all the information of the relational database. This setting does not address our problem because we treat target constraints, we do not limit the interpretation of IRI constructors to only concatenation and DM is fixed i.e., the user cannot define it.

Relational to graph data exchange [Boneva *et al.* 2015] is studied by Boneva et al. and they proposed a setting composed of a relational schema, target constraints and the set of st-tgds expressed with nested regular expressions. Also, the authors proposed a representation of a universal solution that is not a graph but a graph pattern with regular expressions as edge labels. This is a limitation for data exchange where we want to materialize the solution. On the other hand, st-tgds and target tgds from [Boneva *et al.* 2015] are more expressive than constructive st-tgds.

Finally, graph data exchange is studied by Barceló et al. [Barceló *et al.* 2013] and consists of transforming source edge labeled graph to target edge labeled graph. The authors proposed a setting composed of a set of source and target labels; and the set of st-tgds are expressed as conjunctive nested regular expressions over the set of source and target labels. Similar to Boneva et al., the authors proposed a universal solution that is not a graph but a graph pattern. This setting is not applicable to our case because authors did not consider either source or target constraints.

Chapter 3

Consistency

3.1 The opposite side of consistency: inconsistency

In this section, we define an inconsistent data exchange setting and show the necessary sufficient conditions for inconsistency. Also, we present the properties of a core pre-solution related to identifying sufficient conditions for inconsistency. In the sequel of this chapter, we fix a constructive relational to RDF data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$, and we consider that the set of st-tgds Σ_{st} is normalized so that for every st-tgd in Σ_{st}, its right-hand side uses exactly one atom. We precise that an instance I of $\mathscr{R}$ refers to an instance of the relational schema $\mathbf{R}$ where the set of dependencies is empty. Also, we assume a fixed library of IRI constructors $\mathbf{F}$ and we define a set of properties w.r.t. a shapes schema $\mathbf{S} = (\mathscr{T}, \delta)$. We denoted this set by $\mathsf{Prop}_{\mathbf{S}} \subset \mathsf{Iri}$, and it is defined as follows:

$$\mathsf{Prop}_{\mathbf{S}} = \{p \mid \exists T \in \mathscr{T}\,.\, \exists S \in \mathscr{T} \cup \{Lit\}.\, \exists \mu \in \{\mathtt{*}, \mathtt{1}, \mathtt{+}, \mathtt{?}\}.\, p :: S^{\mu} \in \delta(T)\}.$$

We recall that a data exchange setting is *consistent* if every consistent source instance admits a solution. Since inconsistency is the opposite of consistency, we define it as follows:

Definition 3.1.1. A data exchange setting is *inconsistent* if there is a source instance that does not admit a solution.

We illustrate with the following example an inconsistent setting.

Example 3.1.2. We consider the database instance I in Figure 3.1a, the following shape schema

$$\begin{aligned}
\mathsf{ShAddress} &\to \mathsf{ex{:}name} :: Lit^{1}\\
\mathsf{ShInstitute} &\to \mathsf{ex{:}address} :: \mathsf{ShAddress}^{1}\\
\mathsf{ShUniversity} &\to \mathsf{ex{:}address} :: Lit^{*}\\
\mathsf{ShConference} &\to \mathsf{ex{:}name} :: Lit^{1};\\
&\qquad\quad \mathsf{ex{:}year} :: Lit^{1};\\
&\qquad\quad \mathsf{ex{:}place} :: Lit^{1};\\
&\qquad\quad \mathsf{ex{:}sponsor} :: \mathsf{ShInstitute}^{1}
\end{aligned}$$

and the following transformation rules

$$\textit{Conference}(x_1, x_2, x_3, x_4) \Rightarrow \mathit{Triple}(f_{con2iri}(x_2, x_3), \mathsf{ex{:}name}, x_2), \tag{σ_1}$$

$$\textit{Conference}(x_1, x_2, x_3, x_4) \Rightarrow \mathit{Triple}(f_{con2iri}(x_2, x_3), \mathsf{ex{:}year}, x_3), \tag{σ_2}$$

$$\textit{Conference}(x_1, x_2, x_3, x_4) \Rightarrow \mathit{Triple}(f_{con2iri}(x_2, x_3), \mathsf{ex{:}place}, x_4) \tag{σ_3}$$

$$\textit{Conference}(x_1, x_2, x_3, x_4) \Rightarrow \mathsf{ShConference}(f_{con2iri}(x_2, x_3)), \tag{σ_4}$$

$$\textit{Conference}(x_1, x_2, x_3, x_4) \wedge \textit{Sponsor}(x_1, x_5) \Rightarrow \mathit{Triple}(f_{con2iri}(x_2, x_3), \mathsf{ex{:}sponsor}, f_{uni2iri}(x_5)), \tag{σ_5}$$

$$\textit{Sponsor}(x_1, x_2) \Rightarrow \mathsf{ShUniversity}(f_{uni2iri}(x_2)), \tag{σ_6}$$

$$\textit{Sponsor}(x_1, x_2) \wedge \textit{Conference}(x_1, x_3, x_4, x_5) \Rightarrow \mathit{Triple}(f_{uni2iri}(x_2), \mathsf{ex{:}address}, x_5). \tag{σ_7}$$

The function interpretations of the IRI constructors $f_{con2iri}$ and $f_{uni2iri}$ are as follows. The first one concatenates name and year of conference and adds the prefix dblp:. The other one replaces blank spaces with character "–" of the sponsors name.

Let J_0 be the core pre-solution to I w.r.t. the data exchange setting seen in Figure 3.1b. We observe that J_0 has the following facts:

- $\mathsf{ShConference}(\mathsf{dblp{:}icdt2019})$,
- $\mathit{Triple}(\mathsf{dblp{:}icdt2019}, \mathsf{ex{:}place}, \textit{Chile})$, and

- $Triple(\mathsf{dblp{:}icdt2019}, \mathsf{ex{:}place}, \mathit{Peru})$.

and in the shapes schema we see the triple constraint ex:place :: Lit^1 is in the definition of ShConference. This set of facts mentioned above is called a *violation* in J_0 because these facts violate the triple constraint with property ex:place. In fact, this is a source of inconsistency.

Assume that the triple constraint with property label ex:place admits having more than one edge. Thus, the above set of facts is not a violation in J_0. Now remark, that the node univ:university−of−lille is also of type ShInstitute. Therefore, the node should have an edge with ex:address and the target node be of type ShAddress, which is an IRI. This target node also should have an edge with ex:name. We observe that the literal node *"Peru"* is not an IRI. Thus, we add a blank node of type ShAddress and a null literal node. Also, we add an edge with label ex:address from univ:university−of−lille to the blank node and we add an edge with label ex:name from the blank node to the null literal node. These nodes and edges require to be added to J_0 because we want to extend J_0 to be a solution to I w.r.t. $\mathscr{E}$. But it is impossible to fuse the blank node of type ShAddress and the null node of type literal. We call these types *conflicting types*. In fact, this is another source of inconsistency.

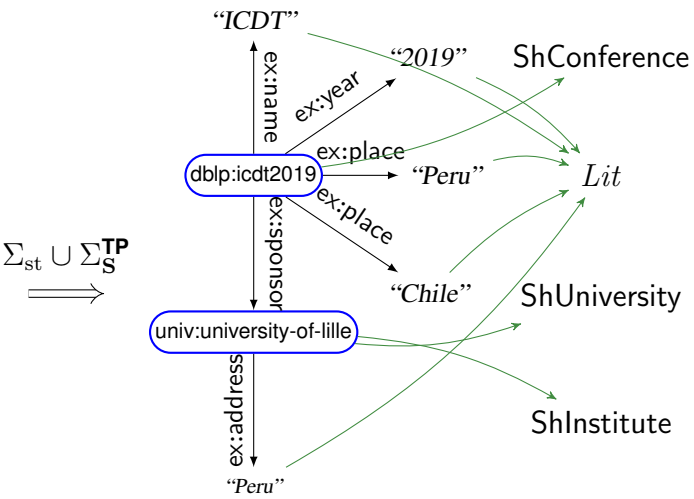
Conference

Idc	*Name*	*Year*	*Place*
1	*ICDT*	2019	*Peru*
2	*ICDT*	2019	*Chile*

Sponsor

Idc	*Name*
1	*University of Lille*

(a) Source instance.



(b) The core pre-solution to I w.r.t. $\mathscr{E}$ of Example 3.1.2.

Figure 3.1: Example of a inconsistency setting.

□

Based on the inconsistency definition, checking inconsistency consists of searching

a source instance for which there is no solution w.r.t. the setting. We recall that a typed graph G is a solution to an instance of $\mathbf{R}$ w.r.t. $\mathscr{E}$, if G satisfies the shapes schema $\mathbf{S}$ and the set of st-tgds Σ_{st}. Also, we recall that a typed graph G is a pre-solution to an instance of $\mathbf{R}$ w.r.t. $\mathscr{E}$, if G satisfies the set of st-tgds Σ_{st}. A graph that together with the source instance of the relational schema satisfies the set of st-tgds is not the source for not obtaining a solution because this graph replicates or transforms the relational data and fits the triple model. But concerning the shapes schema, there are two cases where a graph that together with a source instance I of the relational schema satisfies the set of st-tgds is not a solution to I w.r.t. $\mathscr{E}$:

- If there is a node in the graph for which its outgoing edges with the same predicate are more than one and the shapes schema constraints to be at most one edge.

- If there is a node in the graph for which its outgoing edges with the same predicate are more than one and one of the target nodes is literal and the other is non-literal.

3.1.1 Sources of inconsistency

Now, we define the notion of violation and conflicting types seen in Example 3.1.2 as the sources of inconsistency.

Violation

We define a violation as the property of a graph G.

Definition 3.1.3. A graph G has a violation w.r.t. a shapes schema $\mathbf{S}$ if the following conditions are satisfied:

- There is a type $T \in \mathscr{T}$ for which there is a triple constraint $p :: S^{\mu}$ in its definition for some $S \in \mathscr{T} \cup \{Lit\}$ and some $\mu \in \{\texttt{1}, \texttt{?}\}$

- There are two triples with p where the source node is the same for both triples and $T \in typing(n)$ where n is the source node; and the target nodes are two different constants.

Conflicting types

We define conflicting types as the property of a graph G.

Definition 3.1.4. A typed graph G has conflicting types w.r.t. a shapes schema $\mathbf{S}$ if the following conditions are satisfied:

- there is a type $S \in \mathscr{T} \cup \{Lit\}$, a property label $p \in \mathsf{Prop}_{\mathbf{S}}$, a triple constraint $p :: S^{\mu}$ for some $\mu \in \{1, ?\}$ that is in the shape definition of some type $T \in \mathscr{T}$;
- there are two triples with predicate p in G where the source node n is the same for both triples, $T \in typing(n)$ and one of the target nodes is a null node; and
- if one of the target nodes is typed with S, the other is a literal if S is in $\mathscr{T}$ or not a literal if S is Lit.

3.1.2 Importance of core pre-solution

In Sections 3.2 and 3.3, we are going to focus on core pre-solutions because there are necessary and sufficient conditions that are computed over core pre-solutions. First, we are going to see in Section 3.2 and 3.3 that for a given instance I of $\mathbf{R}$, we can detect sources of inconsistency in the set of pre-solutions to I w.r.t. $\mathscr{E}$, with the help of core pre-solution. Then, we are going to show in Sections 3.2 and 3.3 that this detection of sources of inconsistency are necessary and sufficient conditions for a given instance to not admit a solution. Finally, we do not consider all the instances of $\mathbf{R}$ to verify the consistency of $\mathscr{E}$, but only a finite number of instances whose core pre-solutions present the sources of inconsistency.

3.2 Value consistency

In this section, we define value consistency of the setting $\mathscr{E}$ and we present an analysis of testing value consistency. To define value consistency of $\mathscr{E}$, we start presenting for any instance I of $\mathbf{R}$, the definition of value consistency of a graph w.r.t. I and $\mathscr{E}$ as follows. A graph G is *value consistent* w.r.t. $\mathscr{E}$ and I if $I \cup G \models \Sigma_{\mathrm{st}}$ and $G \models \Sigma_{\mathbf{S}}^{\mathsf{TP}} \cup \Sigma_{\mathbf{S}}^{\mathsf{PF}}$.

Let I be a consistent instance of $\mathbf{R}$. We show that a violation in a core pre-solution to I w.r.t. $\mathscr{E}$ is a sufficient condition to claim that there is no solution to I w.r.t. $\mathscr{E}$.

Lemma 3.2.1. For any instance I of $\mathbf{R}$, if the core pre-solution to I w.r.t. $\mathscr{E}$ is not value consistent, then I does not admit a solution to $\mathscr{E}$.

Proof. Take an instance I of $\mathbf{R}$. We compute its core pre-solution J_0. Assume J_0 is not value consistent. By Definition 2.1.6 of core pre-solution being the unique minimal graph, every solution to I w.r.t. $\mathscr{E}$ contains J_0. Therefore, I does not admit a solution to $\mathscr{E}$. □

Now, we formalize the value consistency of the setting $\mathscr{E}$ as follows.

Definition 3.2.1 (Value consistency)**.** The relational to RDF data exchange setting $\mathscr{E}$ is *value consistent* if for every I instance of $\mathbf{R}$, the core pre-solution to I w.r.t. $\mathscr{E}$ is value consistent.

3.2.1 Testing value consistency

Now, we present an analysis and an algorithm for testing value consistency of $\mathscr{E}$. The analysis consists of

1. preliminary notions that are used to define a finite set of *contentious-based* instances of the signature $\mathscr{R}$,
2. the construction of this finite set denoted by $CInst(\Sigma_{\text{st}})$, and
3. the proof that if there is an instance I of $CInst(\Sigma_{\text{st}})$ that satisfies functional dependencies of $\mathbf{R}$ then there is a violation in the core pre-solution to I w.r.t. $\mathscr{E}$.

This inspection will show that value inconsistency is a sufficient condition for inconsistency of $\mathscr{E}$.

Preliminary notions

Here, we introduce three notions called *violation sort*, *accessibility* and *contentious st-tgds*. We illustrate them with the following example.

Example 3.2.2. Consider the data exchange setting $\mathscr{E}$ of Example 3.1.2. We analyze how do we get the violation W of Example 3.1.2. By definition of violation in a graph, there is a triple constraint with property $p \in \mathsf{Prop_S}$ associated to some type $T \in \mathscr{T}$

definition that forbids two different objects for the same subject typed with T. Since the subject is an IRI and any two IRI constructors have disjoint ranges, then there is only one IRI constructor $f \in \mathscr{F}$ that produces the subject. Thus, every violation is associated with at least one sort (T, f, p). We call the tuple (T, f, p) a *violation sort*.

Consequently, we start computing the set of *violation sorts* w.r.t. $\mathscr{E}$ by combining the shapes ShConference, ShInstitute and ShAddress, with each IRI constructor $f_{con2iri}$ and $f_{uni2iri}$ and with each property $p \in \mathsf{Prop_S}$ such that there is a triple constraint with p and multiplicity `1` or `?`. We obtain the following result.

$$\begin{aligned}\mathscr{V}_s = \{&(\mathsf{ShConference}, f_{con2iri}, \mathsf{ex{:}name}), (\mathsf{ShConference}, f_{con2iri}, \mathsf{ex{:}year}),\\ &(\mathsf{ShConference}, f_{con2iri}, \mathsf{ex{:}place}), (\mathsf{ShConference}, f_{con2iri}, \mathsf{ex{:}sponsor}),\\ &(\mathsf{ShInstitute}, f_{uni2iri}, \mathsf{ex{:}address}), (\mathsf{ShInstitute}, f_{con2iri}, \mathsf{ex{:}address})\\ &(\mathsf{ShConference}, f_{uni2iri}, \mathsf{ex{:}name}), (\mathsf{ShConference}, f_{uni2iri}, \mathsf{ex{:}year}),\\ &(\mathsf{ShConference}, f_{uni2iri}, \mathsf{ex{:}place}), (\mathsf{ShConference}, f_{uni2iri}, \mathsf{ex{:}sponsor})\\ &(\mathsf{ShAddress}, f_{uni2iri}, \mathsf{ex{:}name}), (\mathsf{ShAddress}, f_{con2iri}, \mathsf{ex{:}name})\}\end{aligned}$$

We observe that the violation sort of W is $(\mathsf{ShConference}, f_{con2iri}, \mathsf{ex{:}place})$.

Now in Figure 3.1b, we observe that nodes in the core pre-solution to I w.r.t. $\mathscr{E}$ are generated by the application of rules in Σ_{st}. Also, we observe that the set of st-tgds is normalized, so the head of a rule in Σ_{st} is either a typed atom or a triple atom. We inspect the first two elements of this violation sort $(\mathsf{ShConference}, f_{con2iri})$ by tracing the rules, which are in Σ_{st}, that were triggered such that we can find a node that is the root of a path in the core pre-solution whose last node is constructed by $f_{con2iri}$ and typed with ShConference. Such a sequence of rules is called a *path* in $\mathscr{E}$. The rules of the path are required to be triggered for the existence of the subject node that causes a violation. In our case, the sequence is composed of σ_4 because the triggering of this rule will create a node of typed ShConference produced by the IRI constructor $f_{con2iri}$. We say that the pair $(\mathsf{ShConference}, f_{con2iri})$ is *accessible* in $\mathscr{E}$. We note that a pair can be accessible and not be related to a violation sort. For instance, the pair $(\mathsf{ShUniversity}, f_{uni2iri})$ is accessible in $\mathscr{E}$ but it is not present as part of a violation sort.

Since a rule in $\Sigma_{\mathsf{S}}^{\mathsf{TP}}$ does not produce a node, then the rule σ_3 is triggered twice and

produces two different objects. Because we assume the instance does not contain null values, then the triggering two times rule σ_3 will always produce two different objects. Therefore, if $(\mathsf{ShConference}, f_{con2iri})$ is accessible then rules σ_3 and σ_3 are *contentious* with $(\mathsf{ShConference}, f_{con2iri}, \mathsf{ex{:}place})$. □

Violation sorts. We define the set of violation sorts $\mathscr{V}_s$ w.r.t. $\mathscr{E}$ as the set of triples (T, f, p) w.r.t. $\mathscr{E}$ such that p is a property label for some triple constraint in the definition of T, and f is any function name. Formally, $\mathscr{V}_s = \{(T, f, p) \in \mathscr{T} \times \mathscr{F} \times \mathsf{Prop_S} \mid \exists S \in \mathscr{T} \cup \{Lit\}.\, \exists \mu \in \{\mathtt{1}, \mathtt{?}\}.\, p :: S^{\mu} \in \delta(T)\}$.

In the sequel, we say that G has violation with (T, f, p) if the following set of facts are in G:

$$\{T(f(\mathbf{a})),\, Triple(f(\mathbf{a}), p, b),\, Triple(f(\mathbf{a}), p, b')\}$$

for some tuple of constants $\mathbf{a}$ and some $b, b' \in \mathsf{Lit} \cup \mathsf{Iri}$. Thus, every violation in the core pre-solution has a violation sort.

Accessibility in core pre-solution. We define accessibility, and show that (T, f) is accessible in $\mathscr{E}$ if and only if there is an instance I of the signature $\mathscr{R}$ such that the fact $T(f(\mathbf{a}))$ is in the core pre-solution to I w.r.t. $\mathscr{E}$.

Definition 3.2.3. 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 Σ_{st} if:

- the head of σ_0 is of the form $T_0(f_0(\mathbf{x}_0))$, and
- the head of σ_i is of the form $Triple(f_{i-1}(\mathbf{x}_i), p_i, f_i(\mathbf{y}_i))$ for every $1 \leq i \leq n$, and
- $p_i :: T_i^{\mu_i} \in \delta(T_{i-1})$ for every $1 \leq i \leq n$, and
- $T = T_n$ and $f = f_n$.

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

We call a sequence of rules $\sigma_0, \ldots, \sigma_n$ for n been less or equal than number of rules in Σ_{st}, where each rule is in Σ_{st}, a *path in* $\mathscr{E}$, denoted by π. We point out that π is elementary i.e., in the sequence there are no repetitions of rules. Now, we claim the following lemma.

Lemma 3.2.2. For any $(T, f) \in \mathscr{T} \times \mathscr{F}$, it holds that (T, f) is accessible in $\mathscr{E}$ if and only if there exists an instance I of $\mathscr{R}$ and a tuple of constants $\mathbf{a}$ in the domain of I s.t. the core pre-solution to I w.r.t. $\mathscr{E}$ contains the fact $T(f(\mathbf{a}))$.

Proof. Take $(T, f) \in \mathscr{T} \times \mathscr{F}$. For the $\Rightarrow$ direction. Assume (T, f) is accessible. By definition, there is a path $\pi = \sigma_0, \ldots, \sigma_n$ in $\mathscr{E}$ s.t.:

- $head(\sigma_0) = T_0(f_0(\mathbf{x}_0))$, and
- $head(\sigma_i) = \mathit{Triple}(f_{i-1}(\mathbf{x}_{i-1}), p_i, f_i(\mathbf{y}_i))$ for any $1 \leq i \leq n$, and
- $p :: T_i^{\mu} \in \delta(T_{i-1})$ for some multiplicity μ, and
- $(T, f, p) = (T_n, f_n, p_n)$

for some type symbols $\{T_i \mid 0 \leq i \leq n\} \subseteq \mathscr{T}$, function symbols $\{f_i \mid 0 \leq i \leq n\} \subseteq \mathscr{F}$, IRIs $\{p_i \mid 1 \leq i \leq n\} \subseteq \mathsf{Prop_S}$ and some sequence of variables $\mathbf{x}_i$ and $\mathbf{y}_i$. We construct an instance I of $\mathscr{R}$ such that $adom(I) = \{b\}$ for some $b \in \mathsf{Const}$. We chase I with the sequence $\sigma_0, \ldots, \sigma_n$ and $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$. This chase sequence of st-tgds and $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$ is finite because each rule is applied once. As a result of the chase sequence, there is $\mathbf{b}_i$ for $i \in \{0, \ldots, n\}$ composed of $b \in adom(I)$ such that we have the following result:

$$A = \{T_0(f_0(\mathbf{b}_0)), \mathit{Triple}(f_0(\mathbf{b}_0), p_1, f_1(\mathbf{b}_1)), T_1(f_1(\mathbf{b}_1)), \mathit{Triple}(f_1(\mathbf{b}_1), p_2, f_2(\mathbf{b}_2)), \ldots, \mathit{Triple}(f_{n-1}(\mathbf{b}_{n-1}), p_n, f_n(\mathbf{b}_n)), T_n(f_n(\mathbf{b}_n))\}.$$

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 a homomorphism $h : A \to J$ such that $h(c) = c$ for every $c \in adom(A)$ and J a solution to $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$. Since in A there are no nulls, then $A \subseteq J$. A solution to I w.r.t. $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ is the core pre-solution to I w.r.t. $\mathscr{E}$. Thus, J contains the fact $T_n(f_n(\mathbf{b}_n))$. Because $T = T_n$ and $f = f_n$, it holds that J contains $T(f(\mathbf{b}_n))$.

For the $\Leftarrow$ direction. Assume there exist an instance I of $\mathscr{R}$ and a tuple of constants $\mathbf{a}$ in the domain of I s.t. the core pre-solution to I contains $T(f(\mathbf{a}))$. Since the core pre-solution to I w.r.t. $\mathscr{E}$ can be computed using the chase, then there is a chase sequence of I with $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ such that $T(f(\mathbf{a}))$ is present in the result of the chase sequence. To prove that (T, f) is accessible in $\mathscr{E}$ we require the following claims.

Claim 3.2.2.1. For any I of $\mathscr{R}$, for any finite chase sequence w of I for dependencies $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$, any $a \in \mathsf{Iri}$, any $S \in \mathscr{T}$, any $k \in \{2, \ldots, |w|\}$, if J_k a result of k chase step in w contains the fact $S(a)$ and for any $k' < k$, there is not $S' \in \mathscr{T}$, $p \in \mathsf{Prop}_{\mathbf{S}}$ and $a' \in \mathsf{Iri}$ such that $J_{k'}$ a result of k' chase step contains the facts $S'(a'), Triple(a', p, a)$ then there is $k' < k'' \leq k$, a rule $\sigma \in \Sigma_{\text{st}}$ and a homomorphism $h : \sigma \to J_{k''}$ such that $head(\sigma) = S(f(\mathbf{x}))$ and $h^F(f(\mathbf{x})) = a$ for some $f \in \mathscr{F}$ and some sequence of variables $\mathbf{x}$.

Proof. Assume the premises of the claim. We recall that Σ_{st} is normalized. Because $a \in \mathsf{Iri}$, there is an IRI constructor $f \in \mathscr{F}$ that has generate a. Because a **TP** rule in its terms does not contain an IRI constructor, the generation of a node is done by the application of a rule σ is in Σ_{st}. We recall (1) the restriction of data exchange setting definition in Section 2.1: there is not $T(\mathbf{x})$ or $Lit(f(\mathbf{x}))$ for some $T \in \mathscr{T}$ and some sequence of variable $\mathbf{x}$ in a st-tgd. By σ is in Σ_{st}, by condition that there is not $a' \in \mathsf{Iri}$, $p \in \mathsf{Prop}_{\mathbf{S}}$ and $S' \in \mathscr{T}$ such that $\{S'(a'), Triple(a', p, a)\} \subseteq J_{k'}$, by definition of a chase step that only one rule is triggered and (1), then the head σ if of the form $S(f(\mathbf{x}))$ for some $\mathbf{x} \subseteq vars(\sigma)$ and there is $k' < k'' \leq k$ and a homomorphism $h^F : \sigma \to J_{k''}$ such that $h^F(f(\mathbf{x})) = a$. □

Claim 3.2.2.2. For any finite chase sequence w for $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$, any $k \in \{3, \ldots, |w|\}$, any $b, b' \in \mathsf{Iri}$ and any $q \in \mathsf{Prop}_{\mathbf{S}}$ if J_k a result of k chase step contains the fact $Triple(b, q, b')$ then there is a $k' \leq k$, a rule $\sigma \in \Sigma_{\text{st}}$ such that $head(\sigma) = Triple(f(\mathbf{y_1}), q, g(\mathbf{y_2}))$ for some $f, g \in \mathscr{F}$ and $\mathbf{y_1}$ and $\mathbf{y_2}$ subsets of $vars(\sigma)$; and there is a homomorphism $h^F : \sigma \to J_{k'}$ such that $h^F(\mathbf{y_1}) = b$ and $h^F(\mathbf{y_2}) = b'$.

Proof. Assume J_k contains the fact $Triple(b, q, b')$. Because a triple in a graph is not generated by a **TP** rule and the set of st-tgds Σ_{st} is normalized, then there is a $k' \leq k$ for which a rule $\sigma \in \Sigma_{\text{st}}$ has a homomorphism $h^F : \sigma \to J_{k'}$. The application of a rule in

Σ_{st} that generates a triple is of form $\varphi \Rightarrow \mathit{Triple}(f(\mathbf{y_1}), q, g(\mathbf{y_2}))$ for some $\mathbf{y_1}$ and $\mathbf{y_2}$ subsets of $\mathit{vars}(\varphi)$. Thus, h^F is defined with $h^F(f(\mathbf{y_1})) = b$ and $h^F(f(\mathbf{y_2})) = b'$. $\square$

Now, we fix a chase sequence s such that J_m is the core pre-solution starting at the source instance as follows.

$$I = J_0 \xrightarrow{\sigma_1,h_1} J_1 \xrightarrow{\sigma_2,h_2} \ldots J_{m-1} \xrightarrow{\sigma_m,h_m} J_m$$

where σ_i a dependency in $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ and $h_i^F : \sigma_i \rightarrow J_i$ a homomorphism for any $i \in \{1, \ldots, m\}$.

Let $b = f(\mathbf{a})$. In consequence, the fact $T(b) \in J_m$ has been generated by the application of a rule σ either in Σ_{st} or in $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$ in a instance before J_m. If (a) for any $k' < m$, there is not $S \in \mathscr{T}$, $p \in \mathsf{Prop}_{\mathbf{S}}$, $b' \in \mathsf{Iri}$ such that $J_{k'}$ contains the set of facts $\{S(b'), \mathit{Triple}(b', p, b)\}$ then by Claim 3.2.2.1, there is $k' < k \leq m$, $\sigma \in \Sigma_{\text{st}}$, a homomorphism $h_k^F : \sigma \rightarrow J_k$ such that $\mathit{head}(\sigma) = T(f(\mathbf{x}))$ where f is defined at the beginning and $h_k^F(f(\mathbf{x})) = b$. By head of σ being of the form $T(f(\mathbf{x}))$ and because in definition of accessibility the path can be empty, (T, f) is accessible in $\mathscr{E}$.

Otherwise (b) there is a $k' < k$, $S \in \mathscr{T}$, $p \in \mathsf{Prop}_{\mathbf{S}}$, $b' \in \mathsf{Iri}$ such that $J_{k'}$ contains the set of facts $\{S(b'), \mathit{Triple}(b', p, b)\}$. Also, the rule $S(x) \wedge \mathit{Triple}(x, p, y) \Rightarrow T(y)$ was triggered in a chase step after k' and before k generating the fact $T(b)$. The existence of rule above in $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$ is because there is a triple constraint $p :: T^{\mu} \in \delta(S)$ for some multiplicity μ. By Claim 3.2.2.2 on (b), there is a $k'' \leq k'$, rule $\sigma_1' \in \Sigma_{\text{st}}$ such that its head is of the form $\mathit{head}(\sigma_1') = \mathit{Triple}(f'(\mathbf{x}_1), p, f(\mathbf{x}))$ for some $f' \in \mathscr{F}$ and sequence of variables $\mathbf{x}_1$ and $\mathbf{x}$ and there is a homomorphism $h_{k''}^F : \sigma_1' \rightarrow J_{k''}$ such that $h_{k''}^F(f'(\mathbf{x}_1)) = b'$ and $h_{k''}^F(f(\mathbf{x})) = b$.

Now, we observe the fact $S(b')$ in $J_{k'}$. As done with $T(b)$ in J_m, if condition (a) and because of existence of rule in $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$, then (T, f) is accessible with sequence of rules σ_0', σ_1' where σ_0' is the rule triggered by condition (a). Otherwise condition (b).

Conditions (a) and (b) are evaluated each time an instance contains a fact $T(b)$ for some $T \in \mathscr{T}$ and some $b \in \mathsf{Iri}$. If condition (a) is found, then (T, f) is accessible because each time condition (b) is found, by Claim 3.2.2.2 there is a rule in Σ_{st} that is added to the sequence of rules such that the conditions for accessibility are satisfied. Since there is a start in J_0 and by Claim 3.2.2.1, then the last condition is always (a).

Therefore, (T, f) is accessible in $\mathscr{E}$. □

Contentious st-tgds. We define contentious st-tgds as follows.

Definition 3.2.4. Let (T, f, p) be a violation sort. Two st-tgds $\sigma, \sigma' \in \Sigma_{\text{st}}$ are *contentious* with sort (T, f, p) if

- the head of σ is $\mathit{Triple}(f(\mathbf{z}), p, t)$,
- the head of σ' is $\mathit{Triple}(f(\mathbf{z}'), p, t')$, and
- (T, f) is accessible in $\mathscr{E}$

for some sequence of variables $\mathbf{z}, \mathbf{z}'$ and terms t, t'.

Contentious-based instances

Now, we specify the construction of a finite set of *contentious-based* instances based on the set of violation sorts, accessibility and contentious st-tgds.

Let (T, f, p) be a violation sort, $\sigma, \sigma' \in \Sigma_{\text{st}}$ with sort (T, f, p) be two contentious st-tgds. Let $\pi = \sigma_0, \ldots, \sigma_n$ be a path from which (T, f) is accessible in $\mathscr{E}$, we define $B_{\pi,\sigma,\sigma'}$ as the union of the bodies of the rules that are in the path π together with σ and σ'. We assume that each two rules σ, σ' in π uses mutually disjoint set of variables. For ease of use, we rename σ and σ' with σ_{n+1} and σ_{n+2}. In symbols, the union of the bodies of the rules is $B_{\pi,\sigma,\sigma'} = \bigcup_{i=0}^{n+2} \mathit{body}(\sigma_i)$. Then, we define the sequence of mappings $h_0, \ldots, h_{n+2}$ inductively as follows:

- for any $0 \leq i \leq n+2$, $h_i : \bigcup_{j=0}^{i} \mathit{vars}_{\mathscr{R}}(\sigma_j) \to \mathsf{NullLit}$ is a mapping that is injective when restricted on $\mathit{vars}_{\mathscr{R}}(\sigma_i)$;
- for any $1 \leq i \leq n+2$, h_i coincides with h_{i-1} on the domain of h_{i-1};
- for any $1 \leq i \leq n$, for any head of σ_i of the form $\mathit{Triple}(f_{i-1}(\mathbf{x}_i), p_i, f_i(\mathbf{y}_i))$ for some function symbols f_i, predicate p_i and sequence of variables $\mathbf{x}_i, \mathbf{y}_i$, $h_i(\mathbf{x}_i) = h_{i-1}(\mathbf{y}_{i-1})$ and $h_i(z)$ is fresh w.r.t. the image of h_{n-1} for any $z \notin \mathbf{x}_i$. That is, $z \notin \mathbf{x}_i$ implies $h(z)$ is not in the image of h_{i-1};

- for head of rule σ_{n+1} of the form $Triple(f_n(\mathbf{z}), p, t)$ for some function symbol f_n, sequence of variables $\mathbf{z}$ and some term t, $h_{n+1}(\mathbf{z}) = h_n(\mathbf{y}_n)$ and $h_{n+1}(z)$ is fresh w.r.t. the image of h_n for any $z \notin \mathbf{z}$;

- for head of rule σ_{n+2} of the form $Triple(f_n(\mathbf{z}'), p, t')$ for some function symbol f_n, sequence of variables $\mathbf{z}'$ and some term t', $h_{n+2}(\mathbf{z}') = h_n(\mathbf{y}_n)$ and $h_{n+2}(z)$ is fresh w.r.t. the image of h_{n+1} for any $z \notin \mathbf{z}'$

Now, we define a contentious-based instance denoted by $I_{\pi,\sigma,\sigma'}$ as $I_{\pi,\sigma,\sigma'} = h_{\pi,\sigma,\sigma'}(B_{\pi,\sigma,\sigma'})$ where $h_{\pi,\sigma,\sigma'}$ is a homomorphism $h_{\pi,\sigma,\sigma'} : B_{\pi,\sigma,\sigma'} \to I_{\pi,\sigma,\sigma'}$ defined as $h_{\pi,\sigma,\sigma'} = h_{n+2}$. We illustrate the construction of a contentious-based instance in Example 3.2.5.

Example 3.2.5 (cont. Example 3.1.2 and 3.2.2)**.** We take the following violation sort $(\mathsf{ShInstitute}, f_{uni2iri}, \mathsf{ex{:}address})$ in $\mathscr{V}_s$ of $\mathscr{E}$. We recall that a rule can be contentious with itself. Thus, we take the rule σ_7 and repeat this rule changing its variables. We observe that $(\mathsf{ShInstitute}, f_{uni2iri})$ is accessible with rules σ_4 and σ_5. We rename the variables of rules σ_4, σ_5 and σ_7 and the repeated rule such that two rules use mutually disjoint set of variables.

$\mathit{Conference}(x_1, x_2, x_3, x_4) \Rightarrow \mathsf{ShConference}(f_{con2iri}(x_2, x_3))$

$\mathit{Conference}(x_5, x_6, x_7, x_8) \wedge \mathit{Sponsor}(x_5, x_9) \Rightarrow Triple(f_{con2iri}(x_6, x_7), \mathsf{ex{:}sponsor}, f_{uni2iri}(x_9))$

$\mathit{Sponsor}(y_1, y_2) \wedge \mathit{Conference}(y_1, y_3, y_4, y_5) \Rightarrow Triple(f_{uni2iri}(y_2), \mathsf{ex{:}address}, y_5)$

$\mathit{Sponsor}(y_6, y_7) \wedge \mathit{Conference}(y_6, y_8, y_9, z) \Rightarrow Triple(f_{uni2iri}(y_7), \mathsf{ex{:}address}, z)$

The first step in the construction of contentious-based instances is the union of bodies having as a result:

$$B_{\pi,\sigma,\sigma'} = \{\mathit{Conference}(x_1, x_2, x_3, x_4), \mathit{Conference}(x_5, x_6, x_7, x_8), \mathit{Sponsor}(x_5, x_9), \mathit{Sponsor}(y_1, y_2), \mathit{Conference}(y_1, y_3, y_4, y_5), \mathit{Sponsor}(y_6, y_7), \mathit{Conference}(y_6, y_8, y_9, z)\}.$$

From these atoms in $B_{\pi,\sigma,\sigma'}$, we unify some of the values that is assigned to the variables of $B_{\pi,\sigma,\sigma'}$ in order to get rules σ_4, σ_5, σ_7 and σ_7 been triggered one after the other. For instance, we unify values for variables x_2 and x_6, with distinct value for variables x_3 and

x_7, then variables x_9, y_2 and y_7. Based on these unifications, we define the sequence of homomorphisms h_1, h_2, h_3 and h_4, which are applied to rules above such that h_i coincides in the domain of h_{i-1} for $i \in \{2, 3, 4\}$, as follows.

$$h_1(x_2) = h_2(x_6) = \bot_1 \qquad h_1(x_3) = h_2(x_7) = \bot_2$$
$$h_2(x_9) = h_3(y_2) = h_4(y_7) = \bot_3 \qquad h_3(y_5) = \bot_4 \qquad h_4(z) = \bot_5$$

The rest of variables that occur in rules above are mapped to fresh values. Finally, we apply the homomorphism h_4 to $B_{\pi,\sigma,\sigma'}$ obtaining the contentious-based instance:

$$I_{\pi,\sigma,\sigma'} = \{\textit{Conference}(\bot_6, \bot_{\mathbf{1}}, \bot_{\mathbf{2}}, \bot_7), \textit{Conference}(\bot_8, \bot_{\mathbf{1}}, \bot_{\mathbf{2}}, \bot_9),$$
$$\textit{Sponsor}(\bot_8, \bot_{\mathbf{3}}), \textit{Sponsor}(\bot_{10}, \bot_{\mathbf{3}}), \textit{Conference}(\bot_{10}, \bot_{11}, \bot_{12}, \bot_{\mathbf{4}}),$$
$$\textit{Sponsor}(\bot_{13}, \bot_{\mathbf{3}}), \textit{Conference}(\bot_{13}, \bot_{14}, \bot_{15}, \bot_{\mathbf{5}})\}$$

□

It remains to prove that this instance is unique.

Lemma 3.2.3. $I_{\pi,\sigma,\sigma'}$ is unique up to isomorphism.

Proof. By construction of $I_{\pi,\sigma,\sigma'}$ there is a homomorphism $h_{\pi,\sigma,\sigma'}$ defined from $B_{\pi,\sigma,\sigma'}$ to $I_{\pi,\sigma,\sigma'}$. The values of $I_{\pi,\sigma,\sigma'}$ depend on the actual values given by $h_{\pi,\sigma,\sigma'}$. If we choose different values by a bijective renaming function except for values that are equated in the sequence of mappings defined in the construction, then we obtain an instance that is isomorphic to $I_{\pi,\sigma,\sigma'}$.

We define a bijective rename function for null literal names. Take $(T, f, p) \in \mathscr{V}_s$. To prove that $I_{\pi,\sigma,\sigma'}$ is unique, we assume that there is other contentious-based instance $I'_{\pi,\sigma,\sigma'}$ for contentious rules σ and σ' with sort (T, f, p) such that $I'_{\pi,\sigma,\sigma'} \subset I_{\pi,\sigma,\sigma'}$. By semantics of $\subset$, $I'_{\pi,\sigma,\sigma'}$ contains less facts i.e., there is one rule σ_i for some $i \in \{0, \ldots, n\}$ from the path $\pi = \sigma_0, \ldots, \sigma_n$ in $\mathscr{E}$. But, by definition of contentious, (T, f) is accessible with π and not with π' where π' is the path without the rule σ_i. Thus σ and σ' are not contentious with (T, f, p). Contradiction. □

In the sequel, by $I_{\pi,\sigma,\sigma'}$ we mean an arbitrary instance isomorphic to the one defined above. Finally, we define the set of contentious-based instances as follows:

$$CInst(\Sigma_{\text{st}}) = \{I_{\pi,\sigma,\sigma'} \mid \sigma, \sigma' \in \Sigma_{\text{st}}, \exists (T, f, p) \in \mathscr{V}_s.\ (T, f) \text{ is accessible in } \mathscr{E} \text{ with } \pi \text{ and } \sigma, \sigma' \text{ are contentious with } (T, f, p)\}.$$

We show with the following proposition that for any instance $I$of $\mathscr{R}$, there is a homomorphism from $I_{\pi,\sigma,\sigma'}$ to I if and only if there is a violation in the core pre-solution to I w.r.t. $\mathscr{E}$.

Proposition 3.2.6. For any instance $I$of $\mathscr{R}$, there is a violation sort (T, f, p) such that there exist π, σ, σ', h s.t. (T, f) is accessible with path π in $\mathscr{E}$, σ, σ' are contentious st-tgds with (T, f, p) and $h : I_{\pi,\sigma,\sigma'} \to I$ is a homomorphism if and only if there exist a tuple of constants $\mathbf{a}$ from the domain of I and constants b, b' s.t. the core pre-solution to I w.r.t. $\mathscr{E}$ includes $\{T(f(\mathbf{a})), \mathit{Triple}(f(\mathbf{a}), p, b), \mathit{Triple}(f(\mathbf{a}), p, b')\}$ with $b \neq b'$ and there is triple constraint $p :: S^{\mu} \in \delta(T)$ with some $S \in \mathscr{T} \cup \{Lit\}$ and some $\mu \in \{\mathtt{1}, \mathtt{?}\}$.

Proof. Take an instance I of $\mathscr{R}$. For the $\Rightarrow$ direction. Assume there is a violation sort (T, f, p) and there exists π, σ, σ', h s.t. (T, f) is accessible with path π in $\mathscr{E}$, σ, σ' are contentious st-tgds with (T, f, p) and $h : I_{\pi,\sigma,\sigma'} \to I$ is a homomorphism. Because π is a path for (T, f), then the sequence of st-tgds $\sigma_0, \ldots, \sigma_n$ that composes π is as in Definition 3.2.3 and $T_n = T$ and $f_n = f$. Thus, σ_0 contains a typed atom and for $\sigma_1, \ldots, \sigma_n$ there are triple constraints such that $p_i :: T_i^{\mu_i} \in \delta(T_{i-1})$ for every $1 \leq i \leq n$. Because of existence of these triple constraints, then we have $\Sigma_{\text{S}}^{\mathsf{TP}}$ contains the rules $\mathsf{TP}(T_{i-1}, p_i, T_i)$ for any $0 < i \leq n$. By construction of $I_{\pi,\sigma,\sigma'}$, rules in π and σ, σ' are rewritten by renaming the variables such that two rules use mutually disjoint set of variables and there is a homomorphism $h_{\pi,\sigma,\sigma'} : B_{\pi,\sigma,\sigma'} \to I_{\pi,\sigma,\sigma'}$. For the purpose of defining a sequence of mappings such that the nodes generated by the application of rules of π, σ, σ' are typed following the TP rules, we define that each variable has to be distinct in TP rules and be distinct from those used in $B_{\pi,\sigma,\sigma'}$ as follows. Let $\mathsf{TP}(T_{i-1}, p_i, T_i) = T_{i-1}(u_i) \wedge \mathit{Triple}(u_i, p_i, v_i) \Rightarrow T_i(v_i)$ for any $0 < i \leq n$, where w.l.o.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 $i \neq j$.

Now we define a mapping h' from $B_{\pi,\sigma,\sigma'}$ to I as $h' = h \circ h_{\pi,\sigma,\sigma'}$. It follows by definition of $I_{\pi,\sigma,\sigma'}$ and h 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 $I_{\pi,\sigma,\sigma'}$. Next, we define a chase sequence s starting at I with π, σ, σ' and **TP** rules; 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 with rule σ_0 i.e., $I \xrightarrow{\sigma_0, h'} I_0$, whose head of σ_0 is a typed atom. Let $head(\sigma_0) = T_0(f_0(\mathbf{x}_0))$. For the following chase steps, we intercalate a rule of π and a rule of **TP**. We recall that by renaming of variables done before, the head of σ_i in π is of the form $Triple(f_{i-1}(\mathbf{x}_i), p_i, f_i(\mathbf{y}_i))$, the head of σ is of the form $Triple(f_n(\mathbf{x}_{n+1}), p, t)$ and the head of σ' is of the form $Triple(f_n(\mathbf{x}_{n+2}), p, t')$ for some terms t, t'. We define subsequent chase steps 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 $h_i(u_i) = h'(f_{i-1}(\mathbf{x}_i))$ and $h_i(v_i) = h'(f_i(\mathbf{y}_i))$.

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) $I_0 = I' \cup h'(B_{\pi,\sigma,\sigma'}) \cup T_0(h(f_0(\mathbf{x}_0)))$

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

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

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 h' on I_0 yields $I'_1 = I_0 \cup h'(head(\sigma_1))$, thus (2) holds.

Now from (1) and (2) we know that I'_1 contains the facts

$$T_0(h'(f_0(\mathbf{x}_0))) \text{ and } Triple(h'(f_0(\mathbf{x}_1)), p_1, f_1(\mathbf{y}_1)) = h'(head(\sigma_1)).$$

Recall that by definition, $h_{\pi,\sigma,\sigma'}(\mathbf{x}_0) = h_{\pi,\sigma,\sigma'}(\mathbf{x}_1)$, so also $h'(\mathbf{x}_0) = h'(\mathbf{x}_1)$, thus h_1 is indeed a homomorphism from $T_{i-1}(u_i) \wedge Triple(u_i, p_i, v_i)$ into I'_i and the resulting instance is indeed $I'_i \cup T_i(h'(f_i(\mathbf{y}_i)))$.

The same arguments apply for the induction step for showing that $h' : \sigma_i \to I_{i-1}$ and $h_i : \mathbf{TP}(T_{i-1}, p_i, T_i) \to I'_i$ are homomorphisms with $h(\mathbf{y}_{i-1}) = h(\mathbf{x}_i)$ and $h'(\mathbf{y}_{i-1}) = h'(\mathbf{x}_i)$, 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

- $I_\sigma = I_n \cup \{ Triple(h(f_n(\mathbf{x}_{n+1})), p, h(t))\}$ and
- $I_{\sigma'} = I_\sigma \cup \{Triple(h(f_n(\mathbf{x}_{n+2})), p, h(t'))\}$,

where $h(\mathbf{x}_{n+1}) = h(\mathbf{x}_{n+2}) = h(\mathbf{y}_n)$.

Finally, we consider the core pre-solution computed using chase. Thus, any chase sequence with $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}^{\mathbf{TP}}$ is finite because rules in Σ_{st} does not contain existential variables so there is no fresh values and the nodes are created from constants in the domain of I; and rules in $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$ only type the nodes. Since rules in π are in Σ_{st} and the **TP** rules used for obtaining $I_{\sigma'}$ are part of $\Sigma_{\mathbf{S}}^{\mathbf{TP}}$, then the core pre-solution to I w.r.t. $\mathscr{E}$ contains $I_{\sigma'}$. Therefore, we conclude

$$\{T_n(h'(f_n(\mathbf{y}_n))), Triple(h'(f_n(\mathbf{x}_{n+1})), p, h'(t)), Triple(h'(f_n(\mathbf{x}_{n+2})), p, h'(t'))\} \subseteq J_0.$$

Because of violation sort definition on (T_n, f_n, p), there is a triple constraint $p :: S^\mu \in \delta(T_n)$ for some $S \in \mathscr{T} \cup \{Lit\}$ and some $\mu \in \{\mathtt{1}, \mathtt{?}\}$.

For the $\Leftarrow$ direction. Assume there exist a tuple of constants $\mathbf{a}$ from the domain of I and constants b, b' s.t. the core pre-solution J_0 to I w.r.t. $\mathscr{E}$ includes the set of facts $W = \{T(f(\mathbf{a})), \mathit{Triple}(f(\mathbf{a}), p, b), \mathit{Triple}(f(\mathbf{a}), p, b')\}$ with $b \neq b'$ and there is triple constraint $p :: S^{\mu} \in \delta(T)$ with some $S \in \mathscr{T} \cup \{Lit\}$ and some $\mu \in \{1, ?\}$. By Lemma 3.2.2 on the fact that $T(f(\mathbf{a}))$ is in J_0, (T, f) is accessible with path π in $\mathscr{E}$. Since every triple is generated by the application of a st-tgd, then there are two rules σ and σ' that were triggered to generate $\mathit{Triple}(f(\mathbf{a}), p, b)$ and $\mathit{Triple}(f(\mathbf{a}), p, b')$. Because every violation in J_0 has a violation sort, then $\{T(f(\mathbf{a})), \mathit{Triple}(f(\mathbf{a}), p, b), \mathit{Triple}(f(\mathbf{a}), p, b')\}$ has as sort (T, f, p). Since there is a violation sort (T, f, p) and (T, f) is accessible, then σ and σ' are contentious with (T, f, p).

Now, we construct a contentious-based instance $I_{\pi,\sigma,\sigma'}$ from the contentious st-tgds σ,σ' and the path π and define an injective function $h : I_{\pi,\sigma,\sigma'} \to I$. We show that h is a homomorphism. Because every node in J_0 is generated by the application of a st-tgd, and J_0 contains the violation with sort (T, f, p), then there is a homomorphism from $h' : B_{\pi,\sigma,\sigma'} \to I$. By construction of $I_{\pi,\sigma,\sigma'}$, there is a homomorphism $h_{\pi,\sigma,\sigma'} : B_{\pi,\sigma,\sigma'} \to I_{\pi,\sigma,\sigma'}$. The function h is defined as follows: for any $\perp \in \mathit{adom}(I_{\pi,\sigma,\sigma'})$ there is $a \in \mathit{adom}(I)$ such that $h(\perp) = a$ if there is $x \in \mathit{adom}(B_{\pi,\sigma,\sigma'})$ and $h_{\pi,\sigma,\sigma'}(x) = \perp$ and $h'(x) = a$. It is easy to see that for any $R \in \mathscr{R}$, every $R(\perp_1, \ldots, \perp_n) \in I_{\pi,\sigma,\sigma'}$ where n is arity of R, there is $R(h(\perp_1), \ldots, h(\perp_n)) \in I$. Thus, h is a homomorphism. □

Necessary condition

Now, we show that value consistency of $\mathscr{E}$ is a sufficient condition for consistency by verifying that contentious-based instances are properly consistent instances of $\mathbf{R}$. We express this with the following theorem.

Theorem 3.2.7. *$\mathscr{E}$ is not value consistent if and only if there is an instance $I_{\pi,\sigma,\sigma'}$ in $\mathit{CInst}(\Sigma_{\mathrm{st}})$ such that there is a solution I' for $I_{\pi,\sigma,\sigma'}$ to the set of functional dependencies Σ_{fd} of $\mathbf{R}$ for which $h \circ h_{\pi,\sigma,\sigma'}(t) \neq h \circ h_{\pi,\sigma,\sigma'}(t')$ where h is the unique homomorphism from $I_{\pi,\sigma,\sigma'}$ to I' and t, t' are two different terms such that head of σ is $\mathit{Triple}(f(\mathbf{x}), p, t)$ and the head of σ' is $\mathit{Triple}(f(\mathbf{x}), p, t')$ where σ, σ' are used in the construction of $I_{\pi,\sigma,\sigma'}$.*

Proof. For the $\Rightarrow$ direction. Assume $\mathscr{E}$ is not value consistent. By negation of definition

of value consistency of $\mathscr{E}$, there is an instance I of $\mathbf{R}$ such that the core pre-solution J_0 to I w.r.t. $\mathscr{E}$ is not value consistent. By negation of definition of value consistent of graph and definition of core pre-solution, the core pre-solution does not satisfy the $\Sigma_{\mathbf{S}}^{\mathbf{PF}}$ rules. This means the core pre-solution J_0 has a violation with some sort (T, f, p) in the set of violations sorts w.r.t. $\mathscr{E}$. Because the existence of a violation, we know there is the set of facts $\{T(f(\mathbf{a})), \mathit{Triple}(f(\mathbf{a}), p, b), \mathit{Triple}(f(\mathbf{a}), p, b')\}$ in J_0 with some tuple of constants $\mathbf{a}$ in the domain of I and constants b, b' such that $b \neq b'$. Also, by existence of violation, we know there is a triple constraint $p :: S^{\mu} \in \delta(T)$ for some $S \in \mathscr{T} \cup \{Lit\}$ and some $\mu \in \{\mathtt{1}, \mathtt{?}\}$. Since J_0 is the core pre-solution to I, then there is a function h_1 that maps values from I to J_0. Because of the existence of violation with sort (T, f, p), σ, σ' are contentious with (T, f, p) in Σ_{st} by Proposition 3.2.6 and J_0 is the result of application Σ_{st}, then $h_1(t) = b$ and $h_1(t') = b'$ where t and t' are two different terms such that the head of σ is $\mathit{Triple}(f(\mathbf{x}), p, t)$ and the head of σ' is $\mathit{Triple}(f(\mathbf{x}), p, t')$.

By Proposition 3.2.6, we can construct $I_{\pi,\sigma,\sigma'}$ and there is a homomorphism h' from $I_{\pi,\sigma,\sigma'}$ to I. By construction of $I_{\pi,\sigma,\sigma'}$, we know there is a homomorphism $h_{\pi,\sigma,\sigma'} : B_{\pi,\sigma,\sigma'} \rightarrow I_{\pi,\sigma,\sigma'}$. We chase $I_{\pi,\sigma,\sigma'}$ with Σ_{fd} obtaining the instance I' such that there is a homomorphism $h : I_{\pi,\sigma,\sigma'} \rightarrow I'$. By property of the chase, I' is a universal solution to $I_{\pi,\sigma,\sigma'}$ w.r.t. Σ_{fd}. This means that there is a $h_u : I' \rightarrow I$. Now, we apply h' to I' obtaining an instance I'' and define a function g over the domain of I' as follows: $g(d) = h'(h(\perp))$ such that $h'(\perp) = d$ with $\perp \in \mathit{adom}(I_{\pi,\sigma,\sigma'})$ for every $d \in \mathit{adom}(I)$. This function is well-defined because given two $\perp_1, \perp_2$ either they are fused or they are identities. This means that $h'(\perp_1) = d$ and $h'(\perp_2) = d$ so $h'(h(\perp_1)) = h'(h(\perp_2))$. Then g is a homomorphism from I to I''. We chase I'' with the same chase sequence used for I obtaining the core pre-solution J' to I'' w.r.t. $\mathscr{E}$. Since J' is the core pre-solution to I'', then there is a function h_2 that maps values from I'' to J'. Since we have used the same chase sequence for I and g maps from I to I'', then $h_2 = g \circ h_1$.

We assume by contradiction that $h_2(b) = h_2(b')$. Here, we have that b, b' are the mapped values from terms t and t' in the head of the contentious st-tgds σ and σ'. Since a term can be either a variable or a function term, then we have two cases:

- Let $t = y$ and $t' = y'$. Since $h_2 = g \circ h_1$ and by definition of g, then $h' \circ h_{\pi,\sigma,\sigma'}(y) = h' \circ h_{\pi,\sigma,\sigma'}(y')$. This means that $h(y) = h(y')$. Because I' is a universal solution

to $I_{\pi,\sigma,\sigma'}$ w.r.t. Σ_{fd} and $h_u : I' \to I$, then it is not possible to equate in I' and be different in I. Thus, $I \not\models \Sigma_{\text{fd}}$; a contradiction.

- Let $t = f(\mathbf{y})$ and $t' = f(\mathbf{y}')$. Since the IRI constructors are non-overlapping and in J_0 every IRI comes from an IRI constructor and $h_2 = g \circ h_1$ and by definition of g, then $h' \circ h_{\pi,\sigma,\sigma'}(f(\mathbf{y})) = h' \circ h_{\pi,\sigma,\sigma'}(f(\mathbf{y}'))$. Simplifying, $h' \circ h_{\pi,\sigma,\sigma'}(y) = h' \circ h_{\pi,\sigma,\sigma'}(y')$ for every $y \in \mathbf{y}$ and $y' \in \mathbf{y}'$. This means that $h(y) = h(y')$ for every $y \in \mathbf{y}$ and $y' \in \mathbf{y}'$. Because I' is a universal solution, $I \not\models \Sigma_{\text{fd}}$; a contradiction.

For the $\Leftarrow$ direction. Assume there is an instance $I_{\pi,\sigma,\sigma'} \in \mathit{CInst}(\Sigma_{\text{st}})$ such that there is a solution I' to $I_{\pi,\sigma,\sigma'}$ w.r.t. Σ_{fd} for which $h \circ h_{\pi,\sigma,\sigma'}(t) \neq h \circ h_{\pi,\sigma,\sigma'}(t')$ where $h : I_{\pi,\sigma,\sigma'} \to I'$ is the unique homomorphism and t, t' are two different terms such that head of σ is $\mathit{Triple}(f(\mathbf{x}), p, t)$ and the head of σ' is $\mathit{Triple}(f(\mathbf{x}), p, t')$ where σ, σ' are used in the construction of $I_{\pi,\sigma,\sigma'}$. By definition of the set of contentious-based instances, there is $\pi, \sigma, \sigma', h_{\pi,\sigma,\sigma'}$ s.t. (T, f) is accessible with π in $\mathscr{E}$, σ, σ' are contentious with sort (T, f, p) and $h_{\pi,\sigma,\sigma'}$ is a homomorphism from $B_{\pi,\sigma,\sigma'}$ to $I_{\pi,\sigma,\sigma'}$.

Now, we compute the core pre-solution J_0 to I' w.r.t. $\mathscr{E}$. Since (T, f, p) exists such that (T, f) is accessible with π in $\mathscr{E}$ and σ, σ' are contentious with (T, f, p) and $h : I_{\pi,\sigma,\sigma'} \to I'$ is a homomorphism then by Proposition 3.2.6, there is a triple constraint $p :: T'^{\mu} \in \delta(T)$ for some $T' \in \mathscr{T} \cup \{\mathit{Lit}\}$ and some $\mu \in \{1, ?\}$ and J_0 contains the set of facts

$$\{T(f(\mathbf{a})), \mathit{Triple}(f(\mathbf{a}), p, b), \mathit{Triple}(f(\mathbf{a}), p, b')\}$$

for some tuple of constants $\mathbf{a}$ in the domain of I' and some constants b, b' where $h \circ h_{\pi,\sigma,\sigma'}(\mathbf{x}) = \mathbf{a}, b = h \circ h_{\pi,\sigma,\sigma'}(t)$ and $b = h \circ h_{\pi,\sigma,\sigma'}(t')$. By hypothesis, $b \neq b'$. Thus, J_0 has a violation with this set of facts, and consequently, J_0 is not value consistent. By Lemma 3.2.1, I' does not admit a solution to $\mathscr{E}$. □

Algorithm

Now, we summarize the previous analysis for testing value consistency in the Algorithm 1. The argument of this algorithm is a constructive data exchange setting and the result is a Boolean value. The output is true if the data exchange setting is value consistent. The Algorithm 1 starts computing the set of contentious-based instances $V_{\mathscr{R}}$. Then

we compute a set of consistent contentious-based instances $V_{\mathbf{R}}$ by removing from $V_{\mathscr{R}}$ inconsistent instances. Finally, the algorithm outputs $false$ if $V_{\mathbf{R}}$ is not empty, otherwise $true$.

Algorithm 1: CVC: check value consistency.

Input: a data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$
Output: $true$ if $\mathscr{E}$ is value consistent,$false$ otherwise.

```
1 V_R = {I_{π,σ,σ'} |
    σ, σ' are contentious with (T, f, p) and (T, f) is accessible with π} ;
2 V_R = {I ∈ V_R | I ⊨ Σ_fd};
3 if V_R ≠ ∅ then
4 |   return false;
5 end
6 else
7 |   return true;
8 end
```

3.3 Node kind consistency

In this section, we define node kind consistency and provide a procedure for deciding if a data exchange setting is node kind consistent. To define node kind consistency, we require two auxiliary functions *co-typing* of a data exchange setting and *co-typing* of a graph. In both cases, the function defines a set of set of types. In the case of a data exchange setting $\mathscr{E}$, for a set of types X in the co-typing result it would be possible to have an instance I in which the types of X co-occurs in the typing of a node of every solution to I w.r.t. $\mathscr{E}$. In the case of a graph G, for a set of types X in the co-typing result there is a graph $G' \supset G$ such that G' satisfies a shapes schema and there is a node $n \in nodes(G')$ such that all types of X co-occur in the typing of n. We illustrate co-typing and node kind inconsistency with the following example.

Example 3.3.1. We consider the data exchange setting $\mathscr{E}$ of Example 3.1.2. We are interested in detecting if the setting is node kind inconsistent. For this purpose, we compute the co-typing graph of $\mathscr{E}$ shown in Figure 3.2. A node of this graph is a set of types $A \subseteq \mathscr{T}$ and an edge with label p between a source node and target node represents that there is a type T in the source node such that there is a triple constraint in the shape

definition of T with property p and multiplicity in $\{1, +\}$ for which the target type of this triple constraint is in the target node.

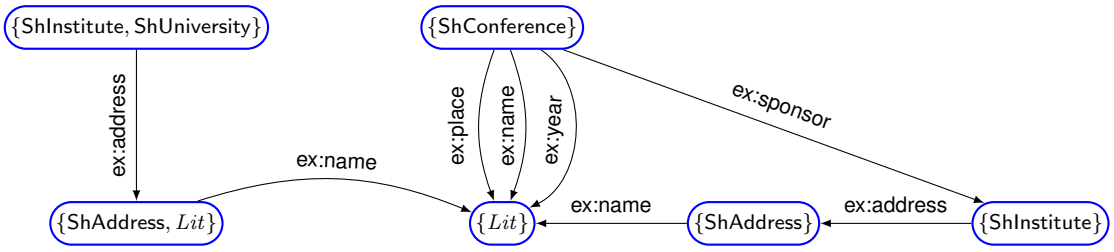


Figure 3.2: Co-typing graph of a setting $\mathscr{E}$.

A co-typing graph provides the following information. Given a node $A \in nodes(G_{\mathscr{E}})$, it would be possible that there is an instance I of $\mathbf{R}$ such that for any solution J to I w.r.t. $\mathscr{E}$ there is a node $n \in nodes(J)$ for which two or more types contained in A co-occurs in the typing of n. For instance, in the co-typing graph seen in Figure 3.2, we take the node $\{\mathsf{ShInstitute}, \mathsf{ShUniversity}\}$. Indeed, for the instance in Figure 3.1a, the core pre-solution seen in Figure 3.1b, which is in every solution to I w.r.t. $\mathscr{E}$, has in its nodes the node univ:university-of-Lille where ShInstitute and ShUniversity co-occurs.

With the information of a co-typing graph, we can compute the co-typing of a graph G that defines a set of set of types where each set of types will probably co-occur in any graph that includes G and satisfies the shapes schema. For instance, the co-typing graph in Figure 3.2 shows that there might exist a node where ShInstitute and ShUniversity will co-occur and this node will need to have an outgoing edge with ex:address and the target node will have types Lit and ShAddress. Indeed, even if we have an edge with ex:address in the core pre-solution we have to add a fresh node of typed ShAddress because according to the co-typing graph there must be an outgoing edge with ex:name from this fresh node to a node of type Lit. This outgoing edge with ex:name cannot be from *"Peru"* node because it is literal. Here, we observe that the fresh node of type ShAddress needs to be fused with the node *"Peru"* because the shapes schema constraints to have at most one edge with ex:address from type ShInstitute. Thus, with the help of co-typing of the core pre-solution, we can identify if there is a conflicting type in any graph that includes the core pre-solution and satisfies shapes schema. Indeed, it will be enough to check if there is a literal and non literal type in a node of the nodes of the co-typing graph. □

3.3.1 Co-typing of a data exchange setting and co-typing graph

Here, we define the co-typing of a data exchange setting and the co-typing graph. To define the co-typing of a data exchange setting, we require two auxiliary functions called *type reachability* and *obligatory property label*. We fix a set of types $\mathscr{T}$. The type reachability function over a $\mathscr{T}$-based shapes schema $\mathbf{S} = (\mathscr{T}, \delta)$ denoted by

$$\Delta : 2^{\mathscr{T}} \times \mathsf{Prop}_{\mathbf{S}} \to 2^{\mathscr{T}}$$

defines a set of types reachable by p-labeled edge from a node that satisfies the set of types X, as follows.

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

The obligatory property label function over a $\mathscr{T}$-based shapes schema $\mathbf{S} = (\mathscr{T}, \delta)$ denoted by $Req : 2^{\mathscr{T}} \to 2^{\mathsf{Prop}_{\mathbf{S}}}$ defines a set of property labels that are required by a set of types X, as follows.

$$Req(X) = \{p \mid \exists T \in X.\, \exists S \in \mathscr{T}.\, \exists \mu \in \{\mathtt{1}, \mathtt{+}\}.\, p :: S^{\mu} \in \delta(T)\ \}.$$

The type accessibility function defines a set of types that are accessible with an IRI constructor $f \in \mathscr{F}$ as follows:

$$Acc(f) = \{T \mid (T, f) \text{ accessible in } \mathscr{E}\}.$$

Based on these two auxiliary functions, we define co-typing of $\mathscr{E}$ as follows:

$$CoTypes(\mathscr{E}) = \bigcup_{i=0}^{\infty} N_i,$$

where $N_0 = \{Acc(f) \mid f \in \mathscr{F}\}$, and $N_i = \{\Delta(X, p) \mid X \in N_{i-1},\ p \in Req(X)\}$ for any $i \geq 1$. We call N_0 the *co-base set of a setting*. We claim the following property of the set N_0 as follows.

Lemma 3.3.1. For any $X \subseteq \mathscr{T}$, $X \in N_0$ iff there is an instance I of $\mathbf{R}$ such that for the core pre-solution J_0 to I w.r.t. $\mathscr{E}$, there is a node $n \in nodes(J_0)$ such that n is typed

with all types of X.

Proof. Take $X \subseteq \mathscr{T}$. For the $\Rightarrow$ direction. We assume $X \in N_0$. Then by definition of N_0, $X = Acc(f)$ for some $f \in \mathscr{F}$. By definition of type accessibility, for any $T \in X$, (T, f) is accessible in $\mathscr{E}$. Take $T \in X$. By Lemma 3.2.2, there is an instance I of $\mathbf{R}$ and tuple of constants $\mathbf{a}$ in the domain of I s.t. the core pre-solution J_0 to I w.r.t. $\mathscr{E}$ contains the fact $T(f(\mathbf{a}))$. Let the domain of I be $adom(I) = \{a\}$. Since the active domain of I is composed of one constant, $J_0 \models \Sigma_{\mathbf{S}}^{\mathsf{TP}}$ and for any $T \in X$, (T, f) is accessible in $\mathscr{E}$, then for all types $T \in X$, we have $T(n) \in J_0$ such that $n = f(\mathbf{a})$ in $nodes(J_0)$. Thus, $types_{J_0}(n) = X$.

For the $\Leftarrow$ direction. We assume there is an instance I of $\mathbf{R}$ such that for the core pre-solution J_0 to I w.r.t. $\mathscr{E}$ there is a node $n \in nodes(J_0)$ s.t. $types_{J_0}(n) = X$. Since $J_0 \models \Sigma_{\text{st}}$, then there is an IRI constructor $f \in \mathscr{F}$ such that $f(\mathbf{a}) = n$ for some vector of constants $\mathbf{a}$. Since $types_{J_0}(n) = X$ and f is used to generate n, then for any $T \in X$, (T, f) is accessible in $\mathscr{E}$. Then $X = Acc(f)$ which fulfills definition of N_0. Consequently $X \in N_0$. □

We point out that this function defines a finite set of set of types because each N_i construct subsets of a finite set of types $\mathscr{T}$ and this process eventually reaches a fix point. Finally, we construct the co-typing graph of $\mathscr{E}$ as follows $G_{\mathscr{E}} = \{\, Triple(X, p, Y) \mid X \in CoTypes(\mathscr{E}) \wedge p \in Req(X) \wedge Y = \Delta(X, p)\}$.

3.3.2 Co-typing of a graph

We define an auxiliary function called *frontier* that defines a set of pairs node property where types of the node requires outgoing edges that are missing w.r.t. a shapes schema $\mathbf{S}$. Formally, the frontier function of a graph G w.r.t. a shapes schema $\mathbf{S}$ is as follows:

$$\mathbb{F}_{\mathbf{S}}(G) = \{(n, p) \mid n \in nodes(G),\ p \in Req(types_G(n)),\ \nexists m \in nodes(G).\ Triple(n, p, m) \in G\}$$

In the following, we introduce the set M_0 called *co-base set of a graph* defined as

$$M_0 = \{X \mid \exists n \in nodes(G).\ types(n) = X\}.$$

Now, the co-typing of a graph G w.r.t. a shapes schema $\mathbf{S}$ is defined as follows.

$$CoTypes(G) = M_0 \cup \bigcup_{i \in \mathbb{N}} N_i^G$$

where $N_i^G = \{\Delta(X, p) \mid X \in N_{i-1}^G, p \in Req(X)\}$ for any $i \geq 1$ and $N_0^G = \{X \in M_0 \mid \exists (n, p) \in \mathbb{F}_{\mathbf{S}}(G), types(n) = X\}$.

Now, we show that co-typing of a setting is included in the union, for every instance I of $\mathbf{R}$, of the co-typing of the core pre-solution to I w.r.t. $\mathscr{E}$.

Lemma 3.3.2. For any relational to RDF data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$, it holds that $CoTypes(\mathscr{E}) \subseteq \bigcup_{I \text{ instance of } \mathbf{R}} CoTypes(J_0)$ where J_0 is the core pre-solution to I w.r.t. $\mathscr{E}$.

Proof. Take a set $X \in CoTypes(\mathscr{E})$. We prove that there is an instance I of $\mathbf{R}$ for which the co-typing of the core pre-solution to I w.r.t. $\mathscr{E}$ contains X. We construct an instance I where the active domain is composed of only one constant i.e., $adom(I) = \{a\}$. We compute the core pre-solution J_0 to I w.r.t. $\mathscr{E}$. Since $J_0 \models \Sigma_{\mathbf{S}}^{\mathsf{TP}}$, then a node n in J_0 has all the accessible types with the IRI constructor used to generate n i.e., $types_{J_0}(n) = Acc(f)$ where $f \in \mathscr{F}$. Based on this fact and that the active domain is only one constant for every node $m \in nodes(J_0)$, $types_{J_0}(m) = Acc(g)$ such that g is used to generate m. If we compute M_0, we obtain the same result. Then $N_0 = \{Acc(f) \mid f \in \mathscr{F}\} = M_0$. Since the definition of $N_0^{J_0}$ takes an element of M_0 and because $N_0 = M_0$, then $N_i^{J_0} = N_i$ for every $i \geq 1$. Then, we conclude that $X \in CoTypes(J_0)$.

□

Also, given an instance I of $\mathbf{R}$, we show a property of an element of the co-typing of the core pre-solution to I w.r.t. $\mathscr{E}$.

Lemma 3.3.3. Let I be an instance of $\mathbf{R}$. Let J_0 be the core pre-solution to I w.r.t. $\mathscr{E}$ such $J_0 \models \Sigma_{\mathbf{S}}^{\mathsf{PF}}$. If $X \in CoTypes(J_0)$ then there is a set of types $Y \in CoTypes(\mathscr{E})$ such that $X \subseteq Y$.

Proof. Let I be an instance of $\mathbf{R}$. Let J_0 be the core pre-solution to I w.r.t. $\mathscr{E}$ such $J_0 \models \Sigma_{\mathbf{S}}^{\mathsf{PF}}$. Assume $X \in CoTypes(J_0)$. By definition of co-typing of a graph, (a) $X \in M_0$ or (b) $X \in \bigcup_{i \in \mathbb{N}} N_i^{J_0}$. For case (a), we know that M_0 is the set of set of

types that co-occur in J_0 and by property of J_0 that every IRI node $n \in nodes(J_0)$ is produced by an IRI constructor i.e., there is $f \in \mathscr{F}$ such that $n = f(\mathbf{a})$ for some vector of constants $\mathbf{a}$. Let $Y = Acc(f)$. By definition of co-typing of setting, we know $Y \in CoTypes(\mathscr{E})$. Since the IRI constructors are non-overlapping and $types_{J_0}(n) = X$, then $X \subseteq Y$. For case (b), the proof goes by induction. For $i = 0$, $X \in N_0^{J_0}$. By definition of N_0^G, there is $(n, p) \in \mathbb{F}_{\mathbf{S}}(J_0)$ such that $X = types_{J_0}(n)$. Since $n \in nodes(J_0)$, there is $f \in \mathscr{F}$ such that $f(\mathbf{a}) = n$ for some vector of constants $\mathbf{a}$. As case (a), there is $Y \in CoTypes(\mathscr{E})$ such that $X \subseteq Y$. Now we assume that the hypothesis holds for $i = k$ for some $k > 1$ such that if $X \in N_i^{J_0}$ then there is $Y \in CoTypes(\mathscr{E})$ such that $X \subseteq Y$. Assume $X \in N_{k+1}^G$. By definition of co-typing of setting, there is $X' \in N_k^{J_0}$ such that $X = \Delta(X', p)$ and $p \in Req(X')$. By induction hypothesis, there is $Y' \in CoTypes(\mathscr{E})$ such that $X' \subseteq Y'$. Since $p \in Req(X')$ and the co-typing graph of the setting must have an edge with p, then there is $Y \in CoTypes(\mathscr{E})$ such that $Y = \Delta(Y', p)$.

We claim that for any $Z, Z' \subseteq \mathscr{T}$, and for any $p \in Req(Z)$, if $Z \subseteq Z'$ then $\Delta(Z, p) \subseteq \Delta(Z', p)$. The proof of this claim is as follows. Assume that $Z \subseteq Z'$. Take $p \in Req(Z)$ and take $T \in \Delta(Z, p)$. By definition type reachability, there is some type $S \in Z$ such that $p :: T^{\mu} \in \delta(S)$ for some $\mu \in \{?, 1, +, *\}$. Since $Z \subseteq Z'$, then $S \in Z'$. Since $p :: T^{\mu} \in \delta(S)$ and $S \in Z'$, then $T \in \Delta(Z', p)$. By this claim, we know that $\Delta(X', p) \subseteq \Delta(Y', p)$. Then $X \subseteq Y$. □

Now, given an instance I of $\mathbf{R}$, we define a $\mathscr{T}$-typed graph called *completed* w.r.t. I and $\mathscr{E}$ using the co-typing of the core pre-solution J_0 to I w.r.t. $\mathscr{E}$. We denote the completed graph w.r.t. I and $\mathscr{E}$ by $G_{\mathscr{E}}(I)$. Let $CoT_{J_0} = CoTypes(J_0) \setminus M_0 \cup N_0^{J_0}$ be the difference of co-typing of J_0 with the co-base set of J_0. For any $X \in CoT_{J_0}$ s.t. $X \subseteq \mathscr{T}$, let n_X be a fresh blank node, i.e. $n_X \in \mathsf{Blank} \setminus adom(J_0)$. For any $X \in CoT_{J_0}$ and $p \in Req(X)$, let $n_{X,p}$ be a fresh null literal, i.e. $n_{X,p} \in \mathsf{NullLit} \setminus adom(J)$. We

define $G_{\mathscr{E}}(I)$ as follows:

$$
\begin{aligned}
G_{\mathscr{E}}(I) = &\{\mathit{Triple}(n,p,n_X) \mid (n,p) \in \mathbb{F}_{\mathbf{S}}(J_0) \wedge X = \Delta(\mathit{types}_{J_0}(n),p) \subseteq \mathscr{T}\} \cup \\
&\{\mathit{Triple}(n,p,n_{X,p}) \mid (n,p) \in \mathbb{F}_{\mathbf{S}}(J_0) \wedge \Delta(\mathit{types}_{J_0}(n),p) = \{\mathit{Lit}\}\} \cup \\
&\{\mathit{Triple}(n_X,p,n_{X'}) \mid X \in \mathit{CoT}_{J_0} \wedge p \in \mathit{Req}(X) \wedge X' = \Delta(X,p) \subseteq \mathscr{T}\} \cup \\
&\{\mathit{Triple}(n_X,p,n_{X,p}) \mid X \in \mathit{CoT}_{J_0} \wedge p \in \mathit{Req}(X) \wedge \Delta(X,p) = \{\mathit{Lit}\}\} \cup \\
&\{T(n_X) \mid X \in \mathit{CoT}_{J_0} \wedge T \in X\}.
\end{aligned}
$$

Lemma 3.3.4. Let I be an instance of $\mathbf{R}$. Let J_0 be the core pre-solution to I w.r.t. $\mathscr{E}$ and $G_{\mathscr{E}}(I)$ the completed graph w.r.t. I and $\mathscr{E}$. The typed graph $J_0 \cup G_{\mathscr{E}}(I)$ is a solution to I w.r.t. $\mathscr{E}$.

Proof. Let $J' = J_0 \cup G_{\mathscr{E}}(I)$. We prove that J' is a solution to I w.r.t. $\mathscr{E}$. Since J_0 is included in J', then $J' \cup I$ satisfies Σ_{st}. By construction of the completed graph w.r.t. I and $\mathscr{E}$, the shapes schema $\mathbf{S}$ is satisfied i.e., $G_{\mathscr{E}}(I) \models \Sigma_{\mathbf{S}}$. Since $G_{\mathscr{E}}(I)$ completes J_0 with nodes and edges that are required by the shapes schema and $G_{\mathscr{E}}(I)$ satisfies $\mathbf{S}$, then $J_0 \cup G_{\mathscr{E}}(I)$ satisfies $\mathbf{S}$. Therefore, J' is a solution to I w.r.t. $\mathscr{E}$. □

3.3.3 Formalization

Given an instance I of $\mathbf{R}$, a graph G is *node kind consistent* w.r.t. $\mathscr{E}$ and I if G is the core pre-solution to I w.r.t. $\mathscr{E}$ and $\mathit{CoTypes}(G)$ does not contain a set X s.t. $\{T, \mathit{Lit}\} \subseteq X$ for some $T \in \mathscr{T}$.

Now, we formalize the node kind consistency of the setting $\mathscr{E}$ as follows:

Definition 3.3.2 (Node kind consistency)**.** The data exchange setting $\mathscr{E}$ is *node kind consistent* if for every I instance of $\mathbf{R}$, the core pre-solution to I w.r.t. $\mathscr{E}$ is node kind consistent.

3.3.4 Necessary condition

Now, we show that co-typing of the setting $\mathscr{E}$ is a necessary and sufficient condition for testing node kind consistency of the setting $\mathscr{E}$.

Theorem 3.3.3. *$\mathscr{E}$ is not node kind consistent if and only if $CoTypes(\mathscr{E})$ contains a set X such that $\{Lit, T\} \subseteq X$ for some type T in $\mathscr{T}$.*

Proof. For the $\Rightarrow$ direction. We assume $\mathscr{E}$ is not node kind consistent. By the opposite of definition of node kind consistency of a setting, there is an instance I of $\mathbf{R}$ for which the co-typing of the core pre-solution to I w.r.t. $\mathscr{E}$ is not node kind consistent. By the opposite of definition of node kind consistency of a graph, the co-types of J_0 contains a set X such that $\{Lit, T\} \subseteq X$ for some T in $\mathscr{T}$. By Lemma 3.3.3, there is $Y \in CoTypes(\mathscr{E})$ such that $X \subseteq Y$. Therefore, we conclude that $CoTypes(\mathscr{E})$ contains a set Y such that $\{Lit, T\} \subseteq Y$.

For the $\Leftarrow$ direction. We assume $CoTypes(\mathscr{E})$ contains a set X such that $\{Lit, T\} \subseteq X$ for some type T in $\mathscr{T}$. By Lemma 3.3.2, there is an instance I of $\mathbf{R}$ such that the core pre-solution to I w.r.t. $\mathscr{E}$ contains X. By negation of definition of node kind consistency of a setting, we conclude that $\mathscr{E}$ is not node kind consistent. □

3.3.5 Algorithm for testing node kind consistency

Here, we present an algorithm for testing node kind consistency. This algorithm has as input a constructive data exchange setting $\mathscr{E}$ and as output a Boolean value. $True$ if $\mathscr{E}$ is node kind consistent, otherwise $false$. We recall the definition of path in a graph. For any graph G, a path π in G is defined by a sequence of labels $p_1 \cdot \ldots \cdot p_k$ for $k \geq 0$ such that for every two consecutive labels there are two edges in G with these labels such that target node for the first label is the source node for the second label. If $k = 0$, we denote by symbol ε, the empty path. We extend the reachability function over a graph G to defines a set of nodes that are reachable with a path in a graph G from a set of nodes in G as follows.

$$\nabla_G(N, \varepsilon) = N,$$
$$\nabla_G(N, \pi \cdot p) = \{n \mid \exists n' \in \nabla_G(N, \pi).\ \mathit{Triple}(n', p, n) \in G\}.$$

First, we describe the Algorithm 2 that constructs the co-typing of $\mathscr{E}$. For every function symbol f in the set $\mathscr{F}$, we use the type accessibility function (line 3) and store the result in the set N_0. We assign N_0 to N and initialize the set N_T that will store the

result of the co-typing of $\mathscr{E}$. Then, we repeat the following process until there is no element of X such that $Req(X) = \emptyset$ and there is no new element to be added in N_T. For every element $X \in N$ and for every property label $p \in Req(X)$, we compute the type reachability with X and p. Then, we store the result of type reachability in set N_s. At the end of this process, we update the set N with N_s. If there is no element of N with a literal and non-literal type, the algorithm returns N_T.

Algorithm 2: Cotypes($\mathscr{E}$)

Input: a data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$
Output: $true$ if $\mathscr{E}$ is node kind consistent, or ($false$) if $\mathscr{E}$ is not

$N_0 = \emptyset$;
for $f \in \mathscr{F}$ **do**
  $N_0 = N_0 \cup \{Acc(f)\}$;
end
$N = N_0$;
$N_T = \emptyset$;
while $\exists X \in N.\, Req(X) \neq \emptyset$ *and* $N \not\subseteq N_T$ **do**
  $N_T = N_T \cup N$;
  $N_s = \emptyset$;
  for $X \in N$ **do**
    for $p \in Req(X)$ **do**
      $Y = \Delta(X, p)$;
      $N_s = N_s \cup \{Y\}$;
    end
  end
  $N = N_s$;
end
return N_T;

Finally, we describe the Algorithm 3 that decides if a data exchange setting $\mathscr{E}$ is node kind consistent. We construct the co-typing graph of $\mathscr{E}$ where $CoTypes(\mathscr{E})$ is defined in Algorithm 2. Then we compute all paths with the function $allpaths$ that returns all possible paths in $G_{\mathscr{E}}$ that are not cycles. Then, for every function symbol f in the set $\mathscr{F}$ and for all ternary combinations of types (T, S, S'), we verify if (S, f) and (S', f) are accessible. If so, we verify that S and S' are in nodes of $G_{\mathscr{E}}$. If so, for every path π in P, we use twice the function reachability with π, one with N and the other with N'. Then, we compare if they are the same and if they contain the type T and Lit. If so, the algorithm returns $false$. Otherwise, the algorithm returns $true$.

Algorithm 3: CKC: Check Node Kind Consistency

Input: a data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$
Output: $true$ if $\mathscr{E}$ is node kind consistent, or ($false$) if $\mathscr{E}$ is not

$G_{\mathscr{E}} = \{(X, p, Y) \mid X \in CoTypes(\mathscr{E}), p \in Req(X), Y = \Delta(X, p)\}$;
$P = allpaths(G_{\mathscr{E}})$;
for $f \in \mathscr{F}$ **do**
  for $(T, S, S') \in \mathscr{T} \times \mathscr{T} \cup \{Lit\} \times \mathscr{T} \cup \{Lit\}$ **do**
    if (S, f) *is accessible in* $\mathscr{E}$ *and* (S', f) *is accessible in* $\mathscr{E}$ **then**
      if $\exists N \in nodes(G_{\mathscr{E}}).\, S \in N$ *and* $\exists N' \in nodes(G_{\mathscr{E}}).\, S' \in N'$ **then**
        use N and N';
        for $\pi \in P$ **do**
          $X = \nabla_{G_{\mathscr{E}}}(N, \pi)$;
          $Y = \nabla_{G_{\mathscr{E}}}(N', \pi)$;
          if $X = Y$ *and* $T \in X$ *and* $Lit \in Y$ **then**
            return $false$;
        end
      end
  end
end
return $true$;

3.4 Deciding consistency

In this section, we show that the two conditions of value consistency and node kind consistency are necessary and sufficient conditions for testing consistency. Then, we show these conditions are decidable, and finally, that the problem of consistency for relational to RDF data exchange setting is decidable. The next lemma establishes that for $\mathscr{E}$ being value consistent and node kind consistent is a sufficient condition for $\mathscr{E}$ to be consistent.

Lemma 3.4.1. Let I be an instance of $\mathbf{R}$. If the core pre-solution to I w.r.t. $\mathscr{E}$ is value consistent and node kind consistent, then I admits a solution to $\mathscr{E}$.

Proof. Take an instance I of $\mathbf{R}$. Assume the core pre-solution J_0 to I w.r.t. $\mathscr{E}$ is value consistent and node kind consistent. We prove that there is a typed graph $U \supseteq J_0$ that is a solution to I w.r.t. $\mathscr{E}$. We recall that J_0 is the unique minimal typed graph J_0 that satisfies the st-tgds Σ_{st} and the $\Sigma_{\mathbf{S}}^{\mathsf{TP}}$ rules for $\mathbf{S}$ (cf. Section 1.5). We compute the completed graph $G_{\mathscr{E}}(I)$ w.r.t. I and $\mathscr{E}$. Let $U = J_0 \cup G_{\mathscr{E}}(I)$. By Lemma 3.3.4, U is a solution. □

We are now ready to establish our main results regarding consistency.

Theorem 3.4.1. *$\mathscr{E}$ is consistent iff $\mathscr{E}$ is value consistent and node kind consistent.*

Proof. For the $\Rightarrow$ direction. We prove by contraposition. Assume $\mathscr{E}$ is not value consistent or $\mathscr{E}$ is not node kind consistent. We have to prove $\mathscr{E}$ is inconsistent.

- When $\mathscr{E}$ is not value consistent. By negation of Definition 3.2.1, there is an instance I of $\mathbf{R}$ such that the core pre-solution J_0 to I w.r.t. $\mathscr{E}$ is not value consistent. Since J_0 is contained in every solution then I does not admit a solution w.r.t. $\mathscr{E}$. Consequently, $\mathscr{E}$ is inconsistent.

- When $\mathscr{E}$ is not node kind consistent then there is a set X in $CoTypes(\mathscr{E})$ such that $\{Lit, T\} \subseteq X$ for some T in $\mathscr{T}$. By Lemma 3.3.2, there is an instance I of $\mathbf{R}$ where the co-typing of its core pre-solution J_0 contains X. We construct an instance I where the active domain is composed of only one constant i.e., $adom(I) = \{a\}$ for some $a \in \mathsf{Const}$. Because the active domain is only one constant, J_0 is value consistent. Since $CoTypes(J_0)$ contains X, then J_0 is not node kind consistent. Since we require J_0 to be value consistent and node kind consistent by Lemma 3.4.1 for I to admit a solution, then because we have that J_0 is value consistent and not node kind consistent, then I does not admit a solution to $\mathscr{E}$. Consequently, $\mathscr{E}$ is inconsistent.

For the $\Leftarrow$ direction. We assume $\mathscr{E}$ is value consistent and node kind consistent. We prove that $\mathscr{E}$ is consistent. We take an instance I of $\mathbf{R}$. By definition of value consistent and node kind consistent, its core pre-solution is value consistent and node kind consistent. By Lemma 3.4.1, we know that there is a solution to I w.r.t. $\mathscr{E}$. Then, we conclude that $\mathscr{E}$ is consistent. □

3.4.1 Decidability

Now, we show that the checking the two conditions for consistency independently is decidable.

Node kind consistency

We show that checking node kind consistency is decidable with the following lemma.

Lemma 3.4.2. Deciding whether $\mathscr{E}$ is node kind consistent is in coNP.

Proof. A certificate for deciding node kind inconsistency is composed of types $T, S, S' \in \mathscr{T}$ and a function symbol $f \in \mathscr{F}$. We use the Algorithm 3 to decide if $\mathscr{E}$ is node kind inconsistent. More precisely, we 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). Finding nodes in $G_{\mathscr{E}}$ that contains S and S' can be tested in polynomial time. Reachability of a node in a graph also can be done in polynomial time. Thus the whole algorithm runs in polynomial time. Consequently, deciding whether $\mathscr{E}$ is node kind consistent is in coNP. □

Value consistency

We show that checking value consistency is decidable with the following lemma.

Lemma 3.4.3. Deciding whether $\mathscr{E}$ is value consistent is in coNP.

Proof. Theorem 3.2.7 implies that $\mathscr{E}$ is value inconsistent if and only if there is an instance $I_{\pi,\sigma,\sigma'}$ in $CInst(\Sigma_{\text{st}})$ such that there is a solution I' for $I_{\pi,\sigma,\sigma'}$ to the set of functional dependencies Σ_{fd} of $\mathbf{R}$. This instance I' 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 the setting is indeed value inconsistent with Algorithm 1. Thus, the Algorithm 1 is a procedure that decides if $\mathscr{E}$ is value inconsistent. It remains to prove that Algorithm 1 runs in polynomial time. By construction of a contentious-based instance, I is constructed in polynomial time. The construction of $V_{\mathscr{R}}$ is in polynomial size because the set of violation sorts $\mathscr{V}_s$ is finite, and the construction of $\mathscr{V}_s$ is in polynomial size of $\mathscr{T}, \mathscr{F}$ and $\mathsf{Prop}_{\mathbf{S}}$; and the construction of each $I_{\pi,\sigma,\sigma'}$ is in polynomial size. Then the construction of $V_{\mathbf{R}}$ is in polynomial time because for any $I \in V_{\mathscr{R}}$, testing if I satisfies functional dependencies of $\mathbf{R}$ can be done with the chase of Σ_{fd}. The chase does not increase the size of the instance, and only a polynomial number of chase steps can be executed before a solution to I w.r.t. Σ_{fd} or a failure is reached. The evaluation of each chase step is polynomial since all bodies of dependencies in Σ_{fd} contain exactly two atoms, thus require to compute a unique join in order to be evaluated. Finally, testing the emptiness of $V_{\mathbf{R}}$ is in polynomial time. Consequently, deciding whether $\mathscr{E}$ is value consistent is in coNP. □

Finally, based on these results we show that checking consistency for $\mathscr{E}$ is decidable with the following theorem.

Theorem 3.4.2 (Complexity of consistency)**.** *Checking consistency of a relational to RDF data exchange setting is coNP-complete.*

Proof. To show that checking consistency is coNP-complete, we show that this problem is in coNP (upper bound) and it is as hard as all problems that are in coNP (lower bound). We first prove that deciding consistency is coNP and then the coNP-hardness.

Upper bound. By Theorem 3.4.1, deciding consistency is equivalent to decide value consistency and node kind consistency. Checking value consistency and node kind consistency is coNP as shown in Lemmas 3.4.3 and 3.4.2 respectively. Therefore, deciding consistency is in coNP.

Lower bound. We recall that complement of SAT is coNP-hard. Therefore, to show coNP-hardness, we reduce from the complement of SAT to deciding consistency. Take any CNF $\varphi = c_1 \wedge \ldots \wedge c_m$ for $m \in \mathbb{N}$, where $c_j = \ell_{j,1} \vee \ldots \vee \ell_{j,k_j}$ for $k_j \in \mathbb{N}$ is a clause over the variables $x_1, \ldots, x_n$ for $n \in \mathbb{N}$. We define a data exchange setting $\mathscr{E}_\varphi = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$ that is inconsistent. The relational schema is defined with the following signature $\mathscr{R} = \{V_{\mathbb{t}}, V_{\mathbb{f}}, R_1, \ldots, R_n\}$. The relational schema consists of the following binary relations (each relation with a single key)

$$V_{\mathbb{t}}(\underline{x}, y), V_{\mathbb{f}}(\underline{x}, y), R_1(\underline{x}, y), \ldots, R_n(\underline{x}, y)$$

The library of IRI constructors $\mathbf{F}$ consists of the following set of IRI constructors

$$\mathscr{F} = \{f_1, \ldots, f_n, f_{n+1}\}$$

and their interpretation F is the concatenation of a number, which is the number of function, and the value of the argument i.e., $f_i(x) =$ "i:" $+ \texttt{str}(x)$. The shapes schema $\mathbf{S}$ consists of the following set of types:

$$\mathscr{T} = \{T_1, \ldots, T_n, T_{n+1}\},$$

and the shape constraints:

$$T_j \to p :: T^*_{j+1} \qquad \text{for } 1 \leq j \leq n \text{ and} \tag{3.1}$$

$$T_{n+1} \to p :: Lit^{\mathsf{1}}. \tag{3.2}$$

The set of st-tgds Σ_{st} are defined as follows. First, we have the two rules:

$$V_{\mathbb{t}}(x, y) \Rightarrow \mathit{Triple}(f_{n+1}(x), p, y) \tag{3.3}$$

$$V_{\mathbb{f}}(x, y) \Rightarrow \mathit{Triple}(f_{n+1}(x), p, y) \tag{3.4}$$

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

$$R_i(x, y) \wedge V_{\mathbb{t}}(x, y) \Rightarrow \mathit{Triple}(f_j(x), p, f_{j+1}(x)) \tag{3.5}$$

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

$$R_i(x, y) \wedge V_{\mathbb{f}}(x, y) \Rightarrow \mathit{Triple}(f_j(x), p, f_{j+1}(x)) \tag{3.6}$$

And finally, we add the following two rules:

$$V_{\mathbb{t}}(x, y) \Rightarrow T_1(f_1(x)) \tag{3.7}$$

$$V_{\mathbb{f}}(x, y) \Rightarrow T_1(f_1(x)) \tag{3.8}$$

We claim that

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

For the $\Rightarrow$ direction. Assume $\varphi \in$ SAT. We take a valuation θ that satisfies φ and construct an instance I_θ as follows. We fix 3 constants c, $\mathbb{t}$, and $\mathbb{f}$. The instance is

$$I_\theta = \{V_{\mathbb{t}}(c, \mathbb{t}), V_{\mathbb{f}}(c, \mathbb{f})\} \cup \{R_i(c, \mathbb{t}) \mid i \in \{1, \ldots, n\},\ \theta(x_i) = \mathit{true}\} \cup \{R_i(c, \mathbb{f}) \mid i \in \{1, \ldots, n\},\ \theta(x_i) = \mathit{false}\}.$$

where n is the number of propositional variables that appear in φ. It is easy to see that

I_θ is consistent of $\mathbf{R}$.

Now the proof goes by induction in the number n of different propositional variables that appears in φ. The base case is that $n = 0$ i.e., φ does not have any propositional variables. By construction of $\mathscr{E}_\varphi$, we have that the relational signature is $\mathscr{R} = \{V_\mathbb{t}, V_\mathbb{f}\}$, the set of IRI constructors is $\mathscr{F} = \{f_1\}$, the set of types $\mathscr{T} = \{T_1\}$ and the set of st-tgds Σ_{st} is composed of rules: (3.3),(3.3),(3.7) and (3.8). The shapes schema has one shape definition for T_1 that is $T_1 \to a :: Lit^1$. The instance is $I_\theta = \{V_\mathbb{t}(c, \mathbb{t}), V_\mathbb{f}(c, \mathbb{f})\}$. We chase I_θ with $\Sigma_{\text{st}} \cup \Sigma_\mathbf{S}$ obtaining a graph J that contains $T_1(f_1(c))$ and the two triples $Triple(f_1(c), p, \mathbb{t})$ and $Triple(f_1(c), p, \mathbb{f})$ which violates the shape constraint on the type T_1. Suppose that (IH) our claim holds for u number of propositional variables: $\varphi' \in$ SAT iff $\mathscr{E}'_\varphi$ is not consistent. Let $\varphi = \varphi' \wedge x_{u+1}$. We construct $\mathscr{E}'_\varphi = (\mathbf{R}', \mathbf{S}', \Sigma'_{\text{st}}, \mathbf{F})$, adding one more relation R_{u+1} to the relational signature in $\mathscr{E}_\varphi$. Also, we add the IRI constructor f_{u+2} to those defined in $\mathscr{E}_\varphi$ and the corresponding type T_{u+2}. Then, we add the triple constraint $p :: T^*_{u+2}$ in the definition of T_{u+1} and the triple constraint $p :: Lit^1$ in the definition of the type T_{u+2}. Finally, we add to the Σ_{st} of $\mathscr{E}_\varphi$ the following rules:

$$V_\mathbb{t}(x, y) \Rightarrow Triple(f_{u+2}(x), p, y)$$
$$V_\mathbb{f}(x, y) \Rightarrow Triple(f_{u+2}(x), p, y)$$
$$R_{u+1}(x, y) \wedge V_\mathbb{t}(x, y) \Rightarrow Triple(f_{u+1}(x), p, f_{u+2}(x)).$$

By IH there is a valuation θ that satisfies φ. Let $\theta'(x_i) = \theta(x_i)$ for $i \in \{1, \ldots, u\}$ and $\theta'(x_{u+1}) = true$. We construct $I'_\theta = I_\theta \cup \{R_{u+1}(c, \mathbb{t})\}$. We chase I'_θ with $\Sigma'_{\text{st}} \cup \Sigma_{\mathbf{S}'}$ obtaining a graph J that contains $T_{u+2}(f_{u+2}(c))$ and the two triples $Triple(f_{u+2}(c), p, \mathbb{t})$ and $Triple(f_{u+2}(c), p, \mathbb{f})$ which violates the shape constraint on the type T_{u+1}.

For the $\Leftarrow$ direction. Assume $\mathscr{E}_\varphi$ is not consistent. By negation of Theorem 3.4.1, $\mathscr{E}_\varphi$ is value inconsistent or node kind inconsistent. It is easy to see that $\mathscr{E}_\varphi$ is node kind consistent because by negation of Theorem 3.3.3, the co-typing of $\mathscr{E}_\varphi$ does not contain $\{Lit, T\}$ for some $T \in \mathscr{T}$. Thus, $\mathscr{E}_\varphi$ is value inconsistent. By negation of Definition 3.2.1, there is an instance I of $\mathbf{R}$ such that the core pre-solution J_0 to I is value inconsistent. By negation of definition of graph to be value consistent, there is a violation in J_0. The only triple constraint that can be violated is $p :: Lit^1$ in the type definition of T_{n+1} where n is the number of propositional variables in φ. Consequently

J_0 contains $T_{n+1}(a_{n+1})$, $\mathit{Triple}(a_{n+1}, p, \mathbb{t})$, and $\mathit{Triple}(a_{n+1}, p, \mathbb{f})$, for some a_{n+1}, $\mathbb{f}$, and $\mathbb{t}$. Naturally, the two triples must be introduced with the rules (3.3) and (3.3), and therefore, there is a constant c such that $a_{n+1} = f_{n+1}(c)$, $V_{\mathbb{t}}(c, \mathbb{t}) \in I$, and $V_{\mathbb{f}}(c, \mathbb{f}) \in I$. Now, the proof goes by induction in the number n of propositional variables as done in the proof of the $\Rightarrow$ direction. It is easy to see with the inductive proof for every $j \in \{1, \ldots, n\}$ we have $T_j(f_j(c)) \in J_0$, $\mathit{Triple}(f_j(c), a, f_{j+1}(c)) \in J$. We observe the triples $\mathit{Triple}(f_j(c), a, f_{j+1}(c))$ can only be added by chase with the use of rules (3.5) and (3.6), 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 of $\mathbf{R}$ then there is not $R_i(c, \mathbb{t})$ and $R_i(c, \mathbb{f})$ in I for $i \in \{1, \ldots, n\}$ or I may have none of the two. We can therefore define the following valuation

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

We show that valuation θ satisfies φ by observing that if for the chase triggers a clause (3.5) or (3.6) that corresponds to some literal ℓ of c_j, then θ satisfies c_j. We finish the proof by observing that the proposed reduction is polynomial. □

3.5 Conclusion

We have studied the problem of checking consistency of a constructive data exchange setting. We have proposed a static analysis tool that decides if a constructive setting is consistent or not. This tool consists of checking conditions that are necessary and sufficient for deciding consistency. These conditions are value consistency and node kind consistency. Both conditions use the core pre-solution to do the analysis. The first condition verifies the presence of violations that is when a node has two edges with the same predicate and the neighbor nodes are constants while the shapes schema constrains the node to have one outgoing edge with that predicate. The second condition verifies that there is no malformed rule such that the application of rules can produce a graph with a node whose type is literal and non-literal. The static analysis tool checks the value consistency by creating counter-examples from the setting. If they are valid counter-

examples, instances that are consistent with the relational schema then the setting is not value consistent. The checking of node kind consistency is done by creating a co-typing graph of the setting and evaluating if there is a node where literal and non-literal types co-occur. Finally, the checking of consistency is coNP-complete.

3.6 Related work

Consistency in the case of relational data exchange is undecidable, and decidable classes usually rely on chase termination ensured by restrictions such as acyclicity, or guarded dependencies, or restrictions on the structure of source instances. Relational to RDF data exchange studied here is a particular case of Relational Data Exchange (except for the IRI constructors, which however do not bring difficulties), therefore all results on relational data exchange apply also to the setting studied here.

De Rougemont et Vielleribière studied the problem of source-consistency in the context of approximate data exchange [de Rougemont & Vieilleribière 2007]. The setting is in the context of trees and words i.e., they transform from a source tree to a target tree, and from a source word to a target word. The mappings are specified by transducers. They did not study the problem of checking consistency of the setting, but, the property of far distance and close distance of an instance to the setting can be adapted to test consistency of our setting. In that sense, the checking tool can construct counter-examples and adapt the property of far and close distance, and if there is an instance that is far from the setting then the setting is not consistent. It remains to be proven if it can be decidable.

On the other hand, we can consider an OBDA system as a data exchange setting because it is composed of source relational schema, an ontology, and a set of mappings. Console et Lenzerini [Console & Lenzerini 2014] studied the case of consistency on an OBDA system. They define two conditions that ensure the consistency of an OBDA system, which are faithfulness and protection. Protection means that if an instance of the relational schema does not satisfy the source constraints then there is no solution to the instance w.r.t. the ontology constraints. Faithfulness means that there is a solution to an instance if the instance is consistent with the relational schema. These conditions are related to the condition of value consistency of a constructive setting in the sense

that both conditions of OBDA and constructive setting checks if the source constraints can cause a conflict in the solution w.r.t. the target schema. Authors show that checking consistency of an OBDA system is undecidable, but for OBDA systems with ontologies written in DL-Lite [Calvanese *et al.* 2005], checking consistency is in NP.

Finally, consistency in the case of XML data exchange was studied by Bojańczyk et al. in [Bojańczyk *et al.* 2013]. Authors called *absolute consistency* and it is defined if for every possible source XML document there is a solution. The complexity of checking absolute consistency is coNEXP. To our knowledge, there are no other studies of consistency in other data exchange contexts.

Chapter 4

Certain query answering

4.1 Motivation and problems

Incomplete information is present in the result of the data exchange from relational to RDF in the presence of shapes schema. Moreover, given a source relational instance, there is an infinite number of solutions and, an answer to a query over a single solution does not provide relevant information. We are interested in answers that are preserved in every solution. Therefore, we focus on the study of certain query answering, which is the computation of good answers i.e., answers present in all solutions, because these good answers provide relevant information from graphs that contain incomplete data.

Certain query answering in the context of data exchange involves two main challenges that are reliable answers and materialization of a good solution [Fagin *et al.* 2005a]. For the first challenge, we consider reliable answers to a query w.r.t. a data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$ and a consistent instance I of $\mathbf{R}$ those answers that are preserved in every solution to I w.r.t. $\mathscr{E}$. For the second challenge, we consider that for any class of queries $\mathscr{Q}$, for any data exchange setting $\mathscr{E}$ and for any instance I of $\mathbf{R}$, a graph G is a *good solution* to I w.r.t. $\mathscr{E}$ and $\mathscr{Q}$ if and only if G is finite and for any query $Q \in \mathscr{Q}$, G preserves certain answers to query Q. Then, computing certain answers depends on the family of queries and the family of data exchange settings.

4.2 Results from existing approaches

In this section, we show how existing results can be easily adapted from relational data exchange and query answering.

4.2.1 Super-weakly acyclic tgds

Marnette [Marnette 2009] introduces the class of *super-weakly acyclic tgds* that has the property of chase termination producing a universal solution, which is used to compute certain answers.

Definition of super-weakly acyclic tgds

Super-weak acyclicity [Marnette 2009] is defined using a unification algorithm and its conditions are based on the study of oblivious chase, which is distinguished from restricted chase (presented in Section 1.3.2) in the sense that the rule is triggered even if the head is already satisfied.

The approach to identify whether a set of tgds is super-weakly acyclic uses the skolemization method that replaces every existentially-quantified variable by a Skolem function (cf. Section 1.3) that depends on all the universally-quantified variables that appear before the existential variable (in the original tgd). Marnette defines the skolemization as follows.

Definition 4.2.1. Given a set of tgds Σ, *skolemization*, denoted by $\mathscr{P}(\Sigma)$, is the function that replaces every tgd $\sigma \in \Sigma$ of the form

$$\varphi(\mathbf{x}, \mathbf{y}) \Rightarrow \exists \mathbf{z}.\, \psi(\mathbf{y}, \mathbf{z})$$

by a tgd

$$\varphi(\mathbf{x}, \mathbf{y}) \Rightarrow \psi(\mathbf{y}, f^{\sigma}_{z_1}(\mathbf{x}), \ldots, f^{\sigma}_{z_n}(\mathbf{x}))$$

where $n = |\mathbf{z}|$ and $f^{\sigma}_{z_j}$ is a fresh function symbol of arity $|\mathbf{x}|$. □

We fix Σ to be a set of tgds and let $\mathscr{P}(\Sigma)$ the skolemization of Σ. We recall technical definitions used by Marnette. A place is a pair $(R(\mathbf{t}), i)$ that indicates the term that appears in the position i of a relational atom $R(\mathbf{t})$ where $R(\mathbf{t})$ is a relational atom of

$\mathscr{P}(\Sigma)$ with some vector of terms $\mathbf{t}$ and $1 \leq i \leq |attrs(R)|$. Given a tgd $\sigma \in \Sigma$ and a existential variable $y \in vars(head(\sigma)) \setminus fvars(head(\sigma))$, we define $Out(\sigma, y)$ as the set of places (called output places) in the head of $\mathscr{P}(\sigma)$ where a function term of the form $f_y^\sigma(\mathbf{t})$ occurs for some vector of terms $\mathbf{t}$. Given a tgd $\sigma \in \Sigma$ and a universal variable x of σ, we define $In(\sigma, x)$ as the set of places (called input places) in the body of σ where x occurs.

Given a set of variable names $V \subseteq \mathscr{V}_\mathscr{Y}$, a substitution θ is a function mapping each $v \in V$ to a finite term $\theta(v)$ build upon constants and function symbols. Two places $(R(\mathbf{t}), i)$ and $(R(\mathbf{t}'), j)$ for some vector of terms $\mathbf{t}$ and $\mathbf{t}'$ are unifiable, in symbols $(R(\mathbf{t}), i) \sim (R(\mathbf{t}'), j)$, iff $i = j$ and there exist two substitutions θ and θ' such that for every $t \in \mathbf{t}, t' \in \mathbf{t}'$, it holds that $\theta(t) = \theta'(t')$. Given two sets of places P and P', we write $P \subseteq P'$ for the containment between two sets of places iff for all $p \in P$ there exists some $p' \in P'$ such that $p \sim p'$. Given a set P of places, we define $Move(\Sigma, P)$ as the smallest set of places P' such that $P \subseteq P'$, and for every tgd of the form $\varphi \Rightarrow \psi$ in $\mathscr{P}(\Sigma)$ and for every universal variable x, if $\Gamma_x(\varphi) \subseteq P'$ then $\Gamma_x(\psi) \subseteq P'$, where $\Gamma_x(\varphi)$ and $\Gamma_x(\psi)$ denote the sets of places in φ and ψ where x occurs respectively.

Given two tgds $\sigma, \sigma' \in \Sigma$, we say that σ triggers σ' in Σ, in symbols $\sigma \rightsquigarrow \sigma'$, iff there exists an existential variable y in the head of σ, and a universal variable x in σ occurring both in the body and the head of σ' such that:

$$In(\sigma', x) \subseteq Move(\Sigma, Out(\sigma, y)).$$

A trigger relation is acyclic if there is no σ that triggers σ. A set of tgds Σ is *super-weakly acyclic* iff the trigger relation is acyclic. We illustrate the evaluation of a set of tgds that is super weakly acyclic in Example 4.2.4.

Weakly-recursive shapes schema

We identify a family of *weakly-recursive* shapes schemas whose set of dependencies is super-weakly acyclic tgds. To define this family of shapes schemas, we define the *dependency graph* of a shapes schema, which is a modification of the shape graph of a shapes schema (cf. Section 1.5.2), as follows.

Definition 4.2.2. Let $\mathbf{S} = (\mathscr{T}, \delta)$ be a shapes schema and $G_\mathbf{S} = (\mathscr{T}, E_\mathbf{S})$ be the shape

graph of $\mathbf{S}$. The dependency graph of $\mathbf{S}$ is the directed graph whose set of nodes is $\mathscr{T}$ and has an edge (T, T') if there is an edge with T and T' in the set of edges of the shape graph i.e., $(T, (p, \mu), T') \in E_{\mathbf{S}}$ for some $p \in \mathsf{Prop}_{\mathbf{S}}$ and some $\mu \in \{1, ?, +, *\}$. There are two kind of edges: (T, T') is a *strong edge* if the edge $(T, (p, \mu), T') \in E_{\mathbf{S}}$ has multiplicity μ in $\{1, +\}$; and (T, T') is a *weak edge* if the edge $(T, (p, \mu), T') \in E_{\mathbf{S}}$ has a multiplicity in $\{*, ?\}$. □

We are now ready to define a kind of shapes schema that guarantees a super-weakly acyclic set of dependencies.

Definition 4.2.3. A shapes schema is *weakly-recursive* if for its dependency graph, every cycle has at least one weak edge. □

For instance, Figure 4.1 shows the dependency graph of shapes schema in Example 1.5.1. We observe that this dependency graph has only one cycle composed at least of one weak edge. Thus, the shapes schema in Example 1.5.1 is weakly-recursive.

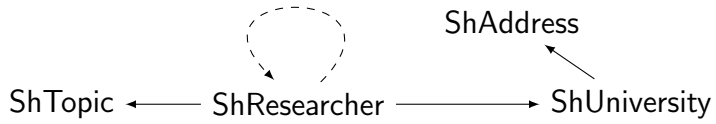


Figure 4.1: Dependency graph of shapes schema in Example 1.5.1.

We recall that a shapes schema can be translated to a set of dependencies as shown in Section 1.5.4. We illustrate that set of dependencies that captures the weakly-recursive shapes schema of Example 1.5.1 is super-weakly acyclic.

Example 4.2.4. Consider the dependency graph shown Figure 4.1 of shapes schema in Example 1.5.1. We consider the following set of tgds $\Sigma_{\mathbf{S}}$ that captures the weak edge and strong edge of the dependency graph. For a strong edge, there is a **PE** and **TP** rule

and for a weak edge only a **TP** rule.

$$\mathsf{ShResearcher}(x) \wedge \mathit{Triple}(x, \mathsf{foaf{:}knows}, y) \Rightarrow \mathsf{ShResearcher}(y)$$

$$\mathsf{ShResearcher}(x) \Rightarrow \exists y.\ \mathit{Triple}(x, \mathsf{ex{:}masters}, y)$$

$$\mathsf{ShResearcher}(x) \wedge \mathit{Triple}(x, \mathsf{ex{:}masters}, y) \Rightarrow \mathsf{ShUniversity}(y)$$

$$\mathsf{ShResearcher}(x) \Rightarrow \exists y.\ \mathit{Triple}(x, \mathsf{ex{:}worksIn}, y)$$

$$\mathsf{ShResearcher}(x) \wedge \mathit{Triple}(x, \mathsf{ex{:}worksIn}, y) \Rightarrow \mathsf{ShUniversity}(y)$$

$$\mathsf{ShUniversity}(x) \Rightarrow \exists y.\ \mathit{Triple}(x, \mathsf{ex{:}address}, y)$$

$$\mathsf{ShUniversity}(x) \wedge \mathit{Triple}(x, \mathsf{ex{:}address}, y) \Rightarrow \mathsf{ShAddress}(y)$$

We skolemize the $\Sigma_{\mathbf{S}}$ obtaining:

$$\sigma_1 : \mathsf{ShResearcher}(x) \wedge \mathit{Triple}(x, \mathsf{foaf{:}knows}, y) \Rightarrow \mathsf{ShResearcher}(y)$$

$$\sigma_2 : \mathsf{ShResearcher}(x) \Rightarrow \mathit{Triple}(x, \mathsf{ex{:}masters}, f_y^{\sigma_2}(x))$$

$$\sigma_3 : \mathsf{ShResearcher}(x) \wedge \mathit{Triple}(x, \mathsf{ex{:}masters}, y) \Rightarrow \mathsf{ShTopic}(y)$$

$$\sigma_4 : \mathsf{ShResearcher}(x) \Rightarrow \mathit{Triple}(x, \mathsf{ex{:}worksIn}, f_y^{\sigma_4}(x))$$

$$\sigma_5 : \mathsf{ShResearcher}(x) \wedge \mathit{Triple}(x, \mathsf{ex{:}worksIn}, y) \Rightarrow \mathsf{ShUniversity}(y)$$

$$\sigma_6 : \mathsf{ShUniversity}(x) \Rightarrow \mathit{Triple}(x, \mathsf{ex{:}address}, f_y^{\sigma_6}(x))$$

$$\sigma_7 : \mathsf{ShUniversity}(x) \wedge \mathit{Triple}(x, \mathsf{ex{:}address}, y) \Rightarrow \mathsf{ShAddress}(y)$$

Then, the set of places in $\Sigma_{\mathbf{S}}$: $p_1 = (\mathsf{ShResearcher}(x), 1), p_2 = (\mathit{Triple}(x, \mathsf{foaf{:}knows}, y), 1), p_3 = (\mathit{Triple}(x, \mathsf{foaf{:}knows}, y), 3), p_4 = (\mathsf{ShResearcher}(y), 1), p_5 = (\mathsf{ShResearcher}(x), 1), \ldots, p_{25} = (\mathsf{ShAddress}(y), 1)$. The set of input places and output places are as follows:

$$\mathit{In}(\sigma_1, y) = \{p_3\} \qquad \mathit{In}(\sigma_2, x) = \{p_5\}$$

$$\mathit{In}(\sigma_3, y) = \{p_{10}\} \qquad \mathit{In}(\sigma_4, x) = \{p_{11}\}$$

$$\mathit{In}(\sigma_5, y) = \{p_{17}\} \qquad \mathit{In}(\sigma_6, x) = \{p_{19}\}$$

$$\mathit{In}(\sigma_7, y) = \{p_{24}\} \qquad \mathit{Out}(\sigma_2, y) = \{p_7\}$$

$$\mathit{Out}(\sigma_4, y) = \{p_{14}\} \qquad \mathit{Out}(\sigma_6, y) = \{p_{21}\}$$

We can observe that the places p_{18} and p_{19} are unifiable because there are two substitutions θ and θ' such that $\theta(x) = \theta'(y)$. Another pair of places that are unifiable are p_7 and p_{10} because there are two substitutions θ and θ' such that $\theta(x, \mathsf{ex{:}masters}, y) = \theta'(x, \mathsf{ex{:}masters}, f_y(x))$. But, places p_2 and p_6 are not unifiable because there is not pair (θ, θ') of substitutions such that $\theta(x, \mathsf{foaf{:}knows}, y) = \theta'(x, \mathsf{ex{:}masters}, y)$. We can check that the only pairs of places that are unifiable are the following: $p_1 \sim p_4, p_6 \sim p_9, p_7 \sim p_{10}, p_{13} \sim p_{16}, p_{14} \sim p_{17}, p_{20} \sim p_{23}, p_{21} \sim p_{24}, p_{18} \sim p_{19}, p_{19} \sim p_{22}, p_4 \sim p_5, p_5 \sim p_8, p_8 \sim p_{12}, p_{12} \sim p_{15}$. Now, for each set of output places we compute the move function(cf. Section 4.2.1):

$$Move(\Sigma_{\mathbf{S}}, Out(\sigma_2, y)) = \{p_7, p_{11}\}$$
$$Move(\Sigma_{\mathbf{S}}, Out(\sigma_4, y)) = \{p_{14}, p_{18}, p_{20}, p_{25}\}$$
$$Move(\Sigma_{\mathbf{S}}, Out(\sigma_6, y)) = \{p_{21}, p_{25}\}$$

Then we compare each two rules $\sigma, \sigma' \in \Sigma_{\mathbf{S}}$ and if for every universal variable in σ' there is an existential variable y in σ such that $In(\sigma', x) \subseteq Move(\Sigma_{\mathbf{S}}, Out(\sigma, y))$, we say that σ triggers σ'. We obtain the following trigger relations: $\sigma_4 \rightsquigarrow \sigma_5$,$\sigma_6 \rightsquigarrow \sigma_7$ and $\sigma_2 \rightsquigarrow \sigma_3$ because

$$\{p_{10}\} = In(\sigma_3, y) \subseteq Move(\Sigma_{\mathbf{S}}, Out(\sigma_2, y)) = \{p_7, p_{11}\}$$
$$\{p_{17}\} = In(\sigma_5, y) \subseteq Move(\Sigma_{\mathbf{S}}, Out(\sigma_4, y)) = \{p_{14}, p_{18}, p_{20}, p_{25}\}$$
$$\{p_{24}\} = In(\sigma_7, y) \subseteq Move(\Sigma_{\mathbf{S}}, Out(\sigma_6, y)) = \{p_{21}, p_{25}\}.$$

We observe that there is no rule σ that triggers itself. Therefore, the set of dependencies $\Sigma_{\mathbf{S}}$ is super-weakly acyclic. □

In general, any cycle with at least one weak edge in the dependency graph of a shapes schema makes trigger relations to be acyclic. Thus, a set of dependencies that captures a weakly recursive schema is super-weakly acyclic as shown in the following lemma.

Lemma 4.2.1. For every shapes schema $\mathbf{S}$, if $\mathbf{S}$ is weakly-recursive then $\Sigma_{\mathbf{S}}$ is super-weakly acyclic.

Proof. Take a shapes schema $\mathbf{S}$. Assume $\mathbf{S}$ is weakly-recursive. We prove by contradiction. Assume $\Sigma_{\mathbf{S}}$ is not super-weakly acyclic i.e., there is $\sigma \in \Sigma_{\mathbf{S}}$ such that $\sigma \rightsquigarrow \sigma$. By definition of trigger relation there is an existential variable y in the head of σ and a universal variable x occurring in body and head of σ such that $In(\sigma, x) \subseteq Move(\Sigma_{\mathbf{S}}, Out(\sigma, y))$. The only kind of rule in $\Sigma_{\mathbf{S}}$ that has an existential variable is a **PE** rule, which also has a universal variable occurring in body and head. Let σ be $T(x) \Rightarrow \exists y.\ Triple(x, p, y)$ for some $T \in \mathscr{T}$. We compute $In(\sigma, x)$ and the function returns a set composed of one position because there is one universal variable occurring in body and head of σ. Let this position be p_i. We compute $Out(\sigma, y)$ and the function returns a set compose of one position because there is only one existential variable. Let this position p_o. The function $Move$ computes positions in heads of rules that are reachable from $Out(\sigma, y)$ such that those positions corresponds to universal variables in rules of the skolemization such that head of σ is unifiable with body of $\sigma' \in \Sigma_{\mathbf{S}}$ in positions p_o and some position in body of σ' and then the head of σ' is unifiable with body of σ'' and so on. Then σ' is a **TP** rule of the form $T(x) \wedge Triple(x, p, y) \Rightarrow T'(y)$ for some $T' \in \mathscr{T}$ and σ'' is a **PE** rule, in essence **TP** and **PE** rules are interleaved. By definition of containment between set of places, there is a position $p_k \in Move(\Sigma_{\mathbf{S}}, Out(\sigma, y))$ such that $p_i \sim p_k$. This means that there are two relational atoms with the same name that are unifiable in positions p_i and p_k. This p_k is in a tgd that corresponds to a **TP** rule such that the head is $T(x)$ and this head is unifiable with the body of σ. Since **PE** corresponds to a multiplicity $\{1, +\}$ and **TP** rule propagates the type and because this unification sequence between **PE** and **TP** rules, then there is a cycle from $T \rightarrow \ldots \rightarrow T$ composed of only strong edges. Thus, $\mathbf{S}$ is not weakly-recursive; a contradiction. □

The property of weakly-acyclic tgds w.r.t. query answering identified by Fagin et al. [Fagin *et al.* 2005a], is also applicable to super-weakly acyclic tgds [Marnette 2009]. Thus, we get the following corollary.

Corollary 4.2.1. *Let $\mathscr{E}$ be a relational to RDF data exchange setting with weakly-recursive target schema. Let I be a consistent instance of $\mathbf{R}$. Let $\mathscr{Q}$ be the class of conjunctive queries over the target schema.*

1. *If the chase with I and $\mathscr{E}$ does not fail then a universal solution J to I w.r.t. $\mathscr{E}$ exists and $cert_{\mathscr{E}}(Q, I) = Q(J)$ for some $Q \in \mathscr{Q}$.*

2. *The data complexity of computing certain answers to a query in $\mathscr{Q}$ w.r.t. I and $\mathscr{E}$ is in PTIME.*

Discussion

This approach of using the result of super-weakly acyclic tgds achieves the two challenges of query answering. A universal solution is materialized and we can decide if a tuple of constants from $\mathsf{Iri} \cup \mathsf{Blank} \cup \mathsf{Lit}$ are certain answers by testing membership in the universal solution. A limitation of this approach is that it only applies to weakly-based settings and not all families of data exchange settings.

4.2.2 Guarded tgds

We recall the definition of guarded tgds. A tgd σ is *guarded* iff it contains an atom in its body that contains all universally quantified variables of σ. We use the results from Calì et al. [Calì *et al.* 2012a] for query answering under guarded tgds.

First, we show that the set of dependencies $\Sigma_{\mathbf{S}}$ that captures the shapes schema is guarded. We recall that $\Sigma_{\mathbf{S}}$ is constructed with the **TP**, **PF** and **PE** rules. We observe that the **TP** rule is guarded because there is a guarded atom that is $Triple$. For the **PE** rule the guarded atom is the type name.

Computing certain answers and data complexity

Based on the results of Calì et al., we can compute certain answers to the family of Boolean conjunctive queries (BCQ) $\mathscr{Q}$ w.r.t. any data exchange setting $\mathscr{E}$. Let I be an instance of $\mathbf{R}$, a query Q in $\mathscr{Q}$ and a data exchange setting $\mathscr{E}$, we construct an auxiliary finite structure using the guarded chase with I and $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}$ as follows. Recall that $\mathscr{G} = \{Triple\}$ is the relational signature of RDF graphs. For every fact in I, we construct all exponential descendants in the guarded chase up to $(|Q| + 1) \cdot |\mathscr{G}| \cdot (2 \cdot w)^w \cdot 2^{|\mathscr{G}| \cdot (2 \cdot w)^w}$ where w is the maximal arity of a relation in $\mathscr{G}$. The complexity of the construction of all descendants is exponential in the size of $\mathscr{R}$. Since the size of the graph signature is one and the value of w is three, then the construction of descendants for a fact in I is up to $(|Q| + 1) \cdot 216 \cdot 2^{216}$. Since $|\Sigma_{\text{st}} \cup \Sigma_{\mathbf{S}}|$ is constant and the guarded chase is seen as a tree, then every node in this tree has only a constant number of children. Thus, the

tree can be constructed in constant time, and the number of applications of tgds in it is constant. Hence, the union of all applications of tgds in the trees of descendants of all $R(\mathbf{a}) \in I$ for some $R \in \mathscr{R}$ and some vector of constants $\mathbf{a}$ can also be constructed in linear time. We compute certain answers to Q w.r.t. I and $\mathscr{E}$ by testing existence of homomorphism of Q in the finite structure. Using the results in data complexity of Calì et al., we claim in the following proposition that data complexity of computing certain answers to BCQs w.r.t. $\mathscr{E}$ is P-complete.

Proposition 4.2.5. For any data exchange setting $\mathscr{E}$ and for any instance I of $\mathbf{R}$, the data complexity of computing certain answers to Boolean conjunctive queries w.r.t. I and $\mathscr{E}$ is P-complete.

Discussion

This approach of using the results of guarded tgds does not achieve the first challenge of materializing a good solution. The second challenge is achieved but the complexity of construction of the finite structure is very high likely because the setting proposed by Calì et al. is more general.

4.3 Simulation-based approach

In this section, we propose a *simulation-based approach* for certain query answering and construct a good solution. This approach covers all constructive data exchange settings that we have defined until now and uses a navigational query language for graphs. Throughout this section, we fix a constructive relational to RDF data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$ and assume it is consistent. We also fix a consistent source instance I of $\mathbf{R}$.

4.3.1 Preliminar notions

Now, we recall two main concepts that are used in the definition of bisimulation.

Graph simulation

The notion of simulation was introduced by Milner [Milner 1971] to define how a process is simulated by another process in a program system. Henzinger et al. [Henzinger *et al.* 1995] adapted the notion of simulation to graphs as the relation between two graphs G and H such for every node $n \in nodes(G)$ there is a node $m \in nodes(H)$ such that every outgoing edge from n has a corresponding outgoing edge from m. In the context of RDF graphs, nodes are labeled and there are two kind of nodes IRIs and literals. Thus, we need to do the following adaptation. Two nodes with different IRIs or literal values are distinguishable and they cannot be simulated one another. Also an IRI cannot simulate a literal and vice-versa. Only blank node can simulate an IRI and null-literal node can simulate a literal value.

Definition 4.3.1. A *simulation* of a graph G by a graph H is a relation

$$R \subseteq nodes(G) \times nodes(H)$$

such that for any $(n, m) \in R$, we have

- n is a literal node if and only if m is a literal node,
- if n is not a null node, then m is not a null node and $n = m$; and
- 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 $(n', m') \in R$. □

A finite family of sets is closed under union if, for any two element of the family, the union of these two sets is in family. It is east to see that any finite family closed under union has exactly one maximal element [Winskel 1993]. The set of simulations of G in H is closed under union, and consequently, there exists exactly one maximal simulation of G in H. When both G and H are known from the context, we denote the maximal simulation of G in H by $\twoheadrightarrow$, we use it as an infix relation symbol, and we say that n is *simulated* by m if $n \twoheadrightarrow m$. Finally, we say that G is *simulated* by H if every node of G is simulated by a node of H.

Nested regular expressions

Nested regular expressions (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 $p \in \mathsf{Prop_S}$, $\ell \in \mathsf{Iri} \cup \mathsf{Lit}$, and $\Box$ is a distinguished wildcard predicate symbol. A NRE E defines a binary relation $[\![E]\!]_G$ on nodes of a graph G as follows.

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

where $[\![E]\!]_G^*$ is the Kleene's closure of $[\![E]\!]_G$ and $[\![E]\!]_G^{-1}$ is the inverse relation of $[\![E]\!]_G$.

An NRE is *forward* if it does not use the inverse operator. In the sequel, we focus on the forward NRE. A forward NRE E is *satisfied* in a graph G, in symbols $G \models E$, if and only if $[\![E]\!]_G \neq \emptyset$. Also, we define the size of a forward NRE as its number of symbols: $|E_1 \cdot E_2| = |E_1 + E_2| = |E_1| + |E_2| + 1, |[E]| = |E^*| = |E| + 1$ and $|\epsilon| = |p| = |\langle \ell \rangle| = |\Box| = 1$.

4.3.2 Forward NRE-based Boolean query language

A query Q can be expressed using forward NREs. We define an answer and certain answer to a query expressed with NRE as follows:

Definition 4.3.2. A pair of nodes (n,m) is an *answer* to a forward NRE E in a graph G iff $(n,m) \in [\![E]\!]_G$. A pair (n,m) is a *certain answer* to a forward NRE E in I w.r.t. $\mathscr{E}$ iff (n,m) is an answer in every solution to I w.r.t. $\mathscr{E}$. □

A forward-based Boolean query is a query expressed by a forward NRE E such that its answers is $true$ if there is an answer to E in G, or $false$ otherwise. $True$ is certain

answer to a forward NRE E in I w.r.t. $\mathscr{E}$ iff $true$ is the answer in every solution to I w.r.t. $\mathscr{E}$.

4.3.3 Robust query classes

Now, we define a class of Boolean queries for which computing certain answers is possible.

Definition 4.3.3. A class $\mathscr{Q}$ of Boolean queries on graphs is *robust under simulation* iff for any query $Q \in \mathscr{Q}$ and any two graphs G and H such that G is simulated by H, if Q is true in G, then Q is true in H. □

Next step, we show that the subclass of forward NREs is robust under simulation.

Lemma 4.3.1. Forward NRE-based Boolean queries are robust under simulation.

Proof. We fix G, H to be graphs and E to be a forward NRE-based Boolean query. We first prove the following claim.

Claim 4.3.1.1. For any two graphs G and H, for any three nodes $n, n' \in nodes(G)$ and $m \in nodes(H)$ if $n \twoheadrightarrow m$ and $(n, n') \in [\![E]\!]_G$ then there is a node $m' \in nodes(H)$ such that $(m, m') \in [\![E]\!]_H$ and $n' \twoheadrightarrow m'$.

Proof. Take two graphs G, H. Take three nodes $n, n' \in nodes(G)$ and $m \in nodes(H)$. Assume $n \twoheadrightarrow m$ and $(n, n') \in [\![E]\!]_G$. The proof is by induction on the structure of E. For a forward NRE E of size $|E| = 1$, we have the following cases:

- When $E = \epsilon$, the assumption is $(n, n') \in [\![\epsilon]\!]_G$. Then $n' = n$ and by assumption $m \in nodes(H)$, trivially $(m, m) \in [\![\epsilon]\!]_H$. Then $m' = m$. Therefore, $n' \twoheadrightarrow m'$.

- When $E = p$, the assumption is $(n, n') \in [\![p]\!]_G$. 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 $E = \Box$, the assumption is $(n, n') \in [\![\Box]\!]_G$. By semantics of $[\![\Box]\!]_G$ and $n \twoheadrightarrow m$, there is a $m' \in nodes(H)$, and $p \in \mathsf{Prop_S}$ such that $(m, p, m') \in H$. Since $(m, p, m') \in H$, then $(m, m') \in [\![p]\!]_H$ and $n' \twoheadrightarrow m'$.

- When $E = \langle \ell \rangle$, the assumption is $(n, n') \in [\![\langle \ell \rangle]\!]_G$. Then $n' = n$ and $n = \ell$. Since $n \twoheadrightarrow m$, then $m = \ell$. By $m = \ell$ and $m \in nodes(H)$, then $(m, m) \in [\![\langle \ell \rangle]\!]_H$. Let $m' = m$, then $n' \twoheadrightarrow m'$.

Now, let E be of size i and assume for any sub expression E' with size less than i, it holds the Claim 4.3.1.1. E can be the composition of two sub expressions E_1 and E_2. Then, we have the following cases:

1. $E = E_1 + E_2$. By semantics of $+$, $(n, n') \in [\![E_1]\!]_G$ or $(n, n') \in [\![E_2]\!]_G$. By induction hypothesis on E_1, there is $m' \in nodes(H)$ such that $(m, m') \in [\![E_1]\!]_H$ and $n' \twoheadrightarrow m'$. Therefore, $(m, m') \in ([\![E_1]\!]_H \cup [\![E_2]\!]_H)$. By previous result, $(m, m') \in [\![E]\!]_H$. The proof for the second case is similar.

2. $E = E_1 \cdot E_2$. By semantics of $\cdot$, $(n, n') \in [\![E_1]\!]_G \circ [\![E_2]\!]_G$. By composition of binary relations, there is $n_2 \in nodes(G)$ such that $(n, n_2) \in [\![E_1]\!]_G$ and $(n_2, n') \in [\![E_2]\!]_G$. By induction hypothesis on $(n, n_2) \in [\![E_1]\!]_G$ and $n \twoheadrightarrow m$, there is $m_2 \in nodes(H)$ such that $(m, m_2) \in [\![E_1]\!]_H$ and $n_2 \twoheadrightarrow m_2$. By induction hypothesis on $n_2 \twoheadrightarrow m_2$ and $(n_2, n') \in [\![E_2]\!]_G$, there is a $m' \in nodes(H)$ such that $(m_2, m') \in [\![E_2]\!]_H$ and $n' \twoheadrightarrow m'$. By $(m, m_2) \in [\![E_1]\!]_H$ and $(m_2, m') \in [\![E_2]\!]_H$ and composition of binary relation, 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 $n' \twoheadrightarrow m'$.

Also, E can be composed of only of one subexpression E_1 as follows:

1. $E = E_1^*$. By semantics of Kleene closure $*$, we have that $(n, n') \in \bigcup_{k \geq 0} [\![E_1]\!]_G^k$. By union of binary relation, we obtain

$$(n, n') \in [\![\epsilon]\!]_G \cup [\![E_1]\!]_G \cup [\![E_1]\!]_G^2 \cup [\![E_1]\!]_G^3 \ldots$$

   By definition of binary relation to some number,

$$(n, n') \in [\![\epsilon]\!]_G \cup [\![E_1]\!]_G \cup [\![E_1]\!]_G \circ [\![E_1]\!]_G \cup [\![E_1]\!]_G \circ [\![E_1]\!]_G^2 \ldots$$

   It was proven for $[\![\epsilon]\!]_G$, that there is $m' \in nodes(H)$ such that $(m, m') \in [\![\epsilon]\!]_H$ and $n' \twoheadrightarrow m'$, consequently, $(m, m') \in [\![E_1]\!]_H^*$. Thus, $(m, m') \in [\![E]\!]_H$. For the

second case, applying the induction hypothesis on E_1, there is a $m' \in nodes(H)$ such that $(m, m') \in [\![E_1]\!]_H$ and $n' \twoheadrightarrow m'$. Thus, $(m, m') \in [\![E]\!]_H$. For the third case, $(n, n') \in [\![E_1]\!]_G \circ [\![E_1]\!]_G$. It was proven for operator $\circ$ that there is $m' \in nodes(H)$ such that $(m, m') \in [\![E_1]\!]_H \circ [\![E_1]\!]_H$ and $n' \twoheadrightarrow m'$, then $(m, m') \in [\![E]\!]_H$. For the fourth case and rest of cases until k, we are going to have the composition of $[\![E_1]\!]_G$ for k times i.e., $[\![E_1]\!]_G \circ [\![E_1]\!]_G \circ [\![E_1]\!]_G \ldots$. We rewrite as $[\![E_1]\!]_G \circ [\![E_2]\!]_G$ where $E_2 = E_{1_1} \cdot E_{1_{k-1}}$. Since E_2 is a sub expression of E and by proof of $\circ$, we have that there is $m' \in nodes(H)$ such that $(m, m') \in [\![E_1]\!]_H \circ [\![E_2]\!]_H$ and $n' \twoheadrightarrow m'$. Thus, $(m, m') \in [\![E]\!]_H$.

2. $E = [E_1]$. By semantics of [], there is an $m'' \in nodes(G)$ and $(n, m'') \in [\![E_1]\!]_G$. Applying induction hypothesis on E_1, there is $m' \in nodes(H)$ such that $(m, m') \in [\![E_1]\!]_H$ and $m'' \twoheadrightarrow m'$. By $(m, m') \in [\![E_1]\!]_H$, $(m, m) \in [\![[E_1]]\!]_H$. Thus, we conclude that $(m, m) \in [\![E]\!]_H$.

□

Next, we show that if $G \models E$ then $H \models E$. Take any two graphs G and H instances of $\mathscr{G}$. 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 $m \in nodes(H)$ such that $n \twoheadrightarrow m$. By Claim 4.3.1.1, $n \twoheadrightarrow m$ and $(n, n') \in [\![E]\!]_G$, there is a node $m' \in nodes(H)$ such that $(m, m') \in [\![E]\!]_H$. Since $[\![E]\!]_H \neq \emptyset$, then $H \models E$. This ends the proof. □

4.3.4 Universal simulation solution

Now, we define the core component of our approach called *universal simulation solution*.

Definition 4.3.4. A universal simulation solution to I w.r.t. $\mathscr{E}$ is a $\mathscr{T}$-typed graph U that is simulated by every solution J to I w.r.t. $\mathscr{E}$.

Construction

We begin the construction of a universal simulation solution to I w.r.t. $\mathscr{E}$ with the core pre-solution J_0 to I w.r.t. $\mathscr{E}$ because J_0 is the unique minimal typed graph that satisfies

st-tgds Σ_{st} and **TP** rules for $\mathbf{S}$ (cf. Section 1.5.4). Then, we compute the completed graph $G_{\mathscr{E}}(I)$ w.r.t. I and $\mathscr{E}$ (cf. Section 3.3.2). Thus, we obtain the typed graph $U = J_0 \cup G_{\mathscr{E}}(I)$. Now, we show that U is a universal simulation solution to I w.r.t. $\mathscr{E}$ with the following lemma.

Lemma 4.3.2. Let I be an instance of $\mathbf{R}$. Let J_0 be the core pre-solution to I w.r.t. $\mathscr{E}$ and $G_{\mathscr{E}}(I)$ be the completed graph w.r.t. I and $\mathscr{E}$. Let $U = J_0 \cup G_{\mathscr{E}}(I)$ be a typed graph. For every solution $J \in sol_{\mathscr{E}}(I)$ to I w.r.t. $\mathscr{E}$, U is simulated by J.

Proof. Take an instance I of $\mathbf{R}$ and a solution $J \in sol_{\mathscr{E}}(I)$. Let J_0 be the core pre-solution to I w.r.t. $\mathscr{E}$ and $G_{\mathscr{E}}(I)$ be the completed graph w.r.t. I and $\mathscr{E}$. We construct the following relation

$$
\begin{aligned}
R = \{(n,m) \in nodes(J_0) \times nodes(J_0) \mid n = m\} \cup \\
\{(n,m) \mid \exists (n_0,p_0) \in \mathbb{F}_{\mathbf{S}}(J_0).\, \exists \pi = p_0 \cdot p_1 \cdot \ldots \cdot p_k. \\
n \in \nabla_U(\{n_0\},\pi) \wedge, m \in \nabla_J(\{n_0\},\pi)\}.
\end{aligned}
$$

where the function ∇_G over a graph and path in a graph are defined in Section 3.3.5.

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

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

To prove the case b, we require the following claim where we use the notion of a path in a graph G and frontier defined in Section 3.3.2. Also, we use the obligatory property label and an extension of the type reachability function presented in Section 3.3.1 to define a set of types reachable by a path as follows:

$$
\begin{aligned}
\Delta^*(X, \pi \cdot p) &= \Delta(\Delta^*(X,\pi), p), \\
\Delta^*(X, \varepsilon) &= X.
\end{aligned}
$$

Claim 4.3.2.1. For any instance I of $\mathbf{R}$, let J_0 be the core pre-solution to I w.r.t. $\mathscr{E}$, let $G_{\mathscr{E}}(I)$ be the completed graph w.r.t. I and $\mathscr{E}$, and let $U = J_0 \cup G_{\mathscr{E}}(I)$. For any solution $J \in sol_{\mathscr{E}}(I)$ to I w.r.t. $\mathscr{E}$ and for any path $\pi \in \mathsf{Prop}^*_{\mathbf{S}}$ in U, it holds that π is also in J.

Proof. Take any instance I of $\mathbf{R}$ and take any solution $J \in sol_{\mathscr{E}}(I)$ to I w.r.t. $\mathscr{E}$. We compute the core pre-solution J_0 to I w.r.t. $\mathscr{E}$. We compute the completed graph $G_{\mathscr{E}}(I)$ w.r.t. I and $\mathscr{E}$. Let $U = J_0 \cup G_{\mathscr{E}}(I)$. Take a path $\pi \in \mathsf{Prop}^*_{\mathbf{S}}$. Assume π exists in U. We prove by induction in the size of π. The base cases is when π is of size 1 and 0 i.e., $\pi = p_0$ and $\pi = \varepsilon$. For the case of size 0, trivially holds. For case of size 1. We distinguish two cases: when π starts at a node that is not in the frontier of J_0 and when π starts at a node that is in the frontier of J_0. For the first case, we know J_0 is in every solution, so π is in J. For the second case where $\pi = p_0$, we know $(n_0, p_0) \in \mathbb{F}_{\mathbf{S}}(J_0)$ and by definition of frontier $p_0 \in Req(types_{J_0}(n_0))$. Since J is a solution to I w.r.t. $\mathscr{E}$ then the outgoing edges required in $\mathbf{S}$ by $types_{J_0}(n_0)$ must be satisfied. Since $p_0 \in Req(types_{J_0}(n_0))$, then there is $m \in nodes(J)$ such that $Triple(n_0, p_0, m) \in J$. Thus, π is in J.

Now, we assume the claim holds for any path π' in U of length $k > 1$. Take any path π in U of length $k+1$. By definition of path in a graph, there is a sequence of nodes $n_0, \ldots, n_k$ such that $(n_{i-1}, p_{i-1}, n_i) \in U$ where $i \in \{1 \ldots, k\}$ and $\pi = p_0 \cdot \ldots \cdot p_{k-1}$. Let $\pi = \pi' \cdot p_{k-1}$ such that $|\pi'| = k-1$ and $\pi' = p_0 \cdot \ldots \cdot p_{k-2}$. By construction of U, we distinguish two cases: when π is in J_0 and since J_0 is included in every solution, then π is in J. The other case is when π is in $G_{\mathscr{E}}(I)$. By hypothesis, we have that π' is a path in J and there are $m_{k-2}, m_{k-1} \in nodes(J)$ such that $Triple(m_{k-2}, p_{k-2}, m_{k-1}) \in J$. By construction of U, every edge in $G_{\mathscr{E}}(I)$ is with a obligatory property label, which means that every solution must have at least one outgoing edge with the obligatory property label. Since J is a solution to I w.r.t. $\mathscr{E}$, then it holds that $Triple(m_{k-1}, p_{k-1}, m_k) \in J$. Thus, π is a path in J. □

We continue the proof of the case b. W.l.o.g., we only consider the case when $n \in nodes(G_{\mathscr{E}}(I))$; the other case is implied by the proof. By $n \in nodes(G_{\mathscr{E}}(I))$ and $(n, m) \in R$, we have that $n \in \nabla_U(\{n_0\}, \pi)$ where $\pi = p_1 \cdot \ldots \cdot p_k$ and $(n_0, p_1) \in \mathbb{F}_{\mathbf{S}}(J_0)$ and $m \in \nabla_J(\{n_0\}, \pi)$. Then, we take $p \in \mathsf{Prop}_{\mathbf{S}}, n' \in nodes(U)$ such that $Triple(n, p, n') \in U$, i.e. $n' \in \nabla_U(\{n_0\}, \pi \cdot p)$ and $\pi \cdot p$ is a path in U. By Claim 4.3.2.1,

we have $\pi \cdot p$ is a path in J i.e., there is a node $m' \in \nabla_J(\{n_0\}, \pi \cdot p)$. Thus, we conclude that $(n', m') \in R$. □

Minimal universal simulation solution

The universal simulation solution as constructed above might have nodes that are equivalent, constructing one without redundancy can be done with the notions of bisimulation and quotient bisimulation. The related notion of bisimulation has found application in normalizing blank nodes and essentially minimizing RDF graphs without altering its informational contents [Tzitzikas *et al.* 2012].

Definition 4.3.5. A *bisimulation* of a graph G is a simulation R of G by G that is symmetric and reflexive.

We note that a non-null node is bisimilar to itself because, by definition of simulation of a graph G, every two non-null nodes $n, m \in nodes(G)$ that are simulated, they are equated $n = m$. Thus, the simulation is reflexive. Because the two non-null nodes are in the same graph, then the simulation is symmetric. Therefore, a non-null node is bisimilar to itself.

Many bisimulations can be defined in a graph. We are interested in the maximal bisimulation because the minimal universal simulation solution is based on the maximal bisimulation. We denote the maximal bisimulation by $\nleftrightarrow$. It is easy to see that bisimulation is an equivalence relation. We fix a graph G and define the equivalence class of a node $n \in nodes(G)$ by $[n]$ and the set of all equivalence classes by $nodes(G)/\nleftrightarrow$. For each equivalence class $C \in nodes(G)/\nleftrightarrow$ we fix an arbitrarily chosen representative node $\eta_C \in C$.

Now, we define the *quotient bisimulation* of a graph G denoted by $G/_{\nleftrightarrow}$ as follows.

Definition 4.3.6. The *quotient bisimulation* of a graph G is the set

$$G/_{\nleftrightarrow} = \{(\eta_{[n]}, p, \eta_{[m]}) \mid (n, p, m) \in G\}.$$

The choice of the representative does not matter in the definition of the quotient because a non-null node is bisimilar only to itself, and consequently, every non-singleton equivalence class in $nodes(G)/\nleftrightarrow$ contains null values only.

Universal simulation solutions are good solutions, but they can contain redundant information. Then, this raises the question if among those good solutions, can we have the best solution. The answer is inspired by the core universal solution proposed in [Fagin *et al.* 2005b] that is based on using minimality as a key criterion for what constitutes the *best* universal solution.

Definition 4.3.7. A *minimal universal simulation solution* U_0 to I w.r.t. $\mathscr{E}$ is a universal simulation solution to I w.r.t. $\mathscr{E}$ such that for any universal simulation solution U to I w.r.t. $\mathscr{E}$, the size of U is greater or equal that the size of U_0.

Now, we show that there is a minimal universal simulation solution to I w.r.t. $\mathscr{E}$. We compute the core pre-solution J_0 to I w.r.t. $\mathscr{E}$. Then, we compute the completed graph $G_{\mathscr{E}}(I)$ w.r.t. I and $\mathscr{E}$. Then, we take the bisimulation quotient of $G_{\mathscr{E}}(I)$ and together with the core pre-solution J_0 we define the $\mathscr{T}$-typed graph U_0 i.e., $U_0 = J_0 \cup G_{\mathscr{E}}(I)/_{\leftrightarrow}$. We point out that because J_0 does not have any null nodes, we can write the $\mathscr{T}$-typed graph U_0 as $U_0 = (J_0 \cup G_{\mathscr{E}}(I))/\!\!\leftrightarrow$. Now, for any universal simulation solution U to I w.r.t. $\mathscr{E}$, we show that U_0 simulated by U.

Lemma 4.3.3. For any universal simulation solution U to I w.r.t. $\mathscr{E}$, U_0 is simulated by U where $U_0 = (J_0 \cup G_{\mathscr{E}}(I))/\!\!\leftrightarrow$ and J_0 is the core pre-solution to I w.r.t. $\mathscr{E}$.

Proof. We compute the core pre-solution J_0 to I w.r.t. $\mathscr{E}$. Take a universal simulation solution U to I w.r.t. $\mathscr{E}$. Let $U' = J_0 \cup G_{\mathscr{E}}(I)$. By Lemma 3.3.4, U' is a solution to I w.r.t. $\mathscr{E}$. By Definition 4.3.4, U is simulated by U'. Let $U_0 = U'/\!\!\leftrightarrow$.

Claim 4.3.3.1. U_0 is simulated by U' and U' is simulated by U_0.

Proof. First, we show that U_0 is simulated by U'. We construct the relation R_0 as follows.

$$R_0 = \{(\eta_{[n]}, m) \in nodes(U_0) \times nodes(U') \mid m \in [n]\}.$$

Take $(\eta_{[n]}, m) \in R_0$. Since nodes of U_0 use the maximal bisimulation of nodes of U'. There is a simulation from nodes U' by nodes of U' that is symmetric and reflexive. Because n also is in U' and $m \in [n]$, then n is simulated by m, and consequently, $\eta_{[n]}$ is simulated by m. Thus, R_0 is a simulation of U_0 by U'.

Now, we show that U' is simulated by U_0. We construct the relation R_1 as follows.

$$R_1 = \{(m, \eta_{[n]}) \in nodes(U') \times nodes(U_0) \mid m \in [n]\}$$

Take $(m, \eta_{[n]}) \in R_0$. Take $m' \in nodes(U')$. Assume $(m, p, m') \in U'$. Since also $n \in nodes(U')$ and $m \in [n]$, then there is $n' \in nodes(U')$ such that $(n, p, n') \in U'$. Thus, $m' \in [n']$ and we have $(m', \eta_{[n']}) \in R_1$. Therefore, U_0 is simulated by U' and U' is simulated by U_0. Now, we show a property of U_0. □

Finally, we can state that U_0 is simulated by U because of Claim 4.3.3.1, the transitivity of simulation, and the following claim.

Claim 4.3.3.2. U' is simulated by U.

Proof. We show that U' is simulated by U by contradiction. Given two nodes $n \in nodes(U_0)$ and $m \in nodes(U)$, we assume that n is not simulated by m. By negation of simulation definition, we negate the three conditions. For the first condition, we have two cases either (1.1) n is literal and m is not literal or (1.2) n is not literal and m is literal. For the first case and second case, there is a contradiction because U is simulated by U' i.e., m is simulated by n, then m is literal if and only if n is literal. For the second condition we have n is not null node and $n \neq m$. Because $n \neq m$, if m is literal and by the first condition proved, we have n is literal; a contradiction.

For the third condition, for any outgoing edge $(n, p, n') \in U'$ either there is no outgoing edge from m with p in U or if there exists a corresponding outgoing edge with $(m, p, m') \in U$ then n' is not simulated by m'. Recall that $U' = J_0 \cup G_{\mathscr{E}}(I)$. Assume $(n, p, n') \in U'$. We have the following cases:

- If $n \in J_0$, then , by second condition proved, m is in J_0 of U. Because $m = n$, there is a $m' \in nodes(U)$ and a corresponding edge $(m, p, m') \in U$; a contradiction with the no existence of edge with p.

- If $n' \in nodes(J_0)$ and n' is not null node then $n' \twoheadrightarrow m'$. Recall that $\mathscr{E}$ is consistent, thus there is no co-occurrence of literal and non literal types for a node. If n' is literal then m' is literal. Because these nodes are literals, there are no outgoing edges, and consequently, $n' \twoheadrightarrow m'$; a contradiction.

- When n is a null node, then $n \in nodes(G_{\mathscr{E}}(I))$. By construction of $G_{\mathscr{E}}(I)$, every edge in $G_{\mathscr{E}}(I)$ corresponds to an obligatory property that must be satisfied in any solution to I w.r.t. $\mathscr{E}$. Since U is simulated by a solution and p is a obligatory property label and any solution must satisfy an obligatory property, then there is $m' \in nodes(U)$ such that $(m, p, m') \in U$; a contradiction with the no existence of edge with p.

- When there is a corresponding outgoing edge with $(m, p, m') \in U$, then because from n' the outgoing edges are those that are required to stay in a solution, then U contains corresponding edges from m'. Consequently, $n' \nrightarrow m'$; a contradiction with n' is not simulated by m'.

□

This ends the proof. □

By transitivity of simulation, U_0 is simulated by every solution J to I w.r.t. $\mathscr{E}$. Therefore, U_0 is a universal simulation solution. Now, we show that U_0 is minimal.

Lemma 4.3.4. Let I be an instance of $\mathbf{R}$. Let J_0 be the core pre-solution to I w.r.t. $\mathscr{E}$ and $G_{\mathscr{E}}(I)$ be the completed graph w.r.t. I and $\mathscr{E}$. Let $U_0 = (J_0 \cup G_{\mathscr{E}}(I))/\!\!\leftrightarrows$. U_0 is the minimal universal simulation solution.

Proof. Let J_0 be the core pre-solution to I w.r.t. $\mathscr{E}$ and $G_{\mathscr{E}}(I)$ the completed graph w.r.t. I and $\mathscr{E}$. Let $U_0 = (J_0 \cup G_{\mathscr{E}}(I))/\!\!\leftrightarrows$. Take any universal simulation solution U to I w.r.t. $\mathscr{E}$ and create an injective mapping from the nodes of U_0 to the nodes of U. The mapping is an identity on J_0 which is contained in any solution. Now, for a node $n \in nodes(G_{\mathscr{E}}(I)/\!\!\leftrightarrows)$ we observe that there must be at least one path π from a frontier node n_0 to n, and because U is simulated by U_0, there exists at least one node $m \in nodes(U)$ such that is reachable from n_0 by path π. Consequently, we map n to an arbitrary such m.

Now, suppose by contradiction that two different nodes $n_1, n_2 \in nodes(G_{\mathscr{E}}(I)/\!\!\leftrightarrows)$ are mapped to the same node m. Because U_0 is a bisimulation quotient and the nodes n_1 and n_2 are different, they are not bisimilar. However, since U_0 is simulated by U, and vice versa, and n_1 is reachable with the same path in U_0 as m in U and n_1 is reachable

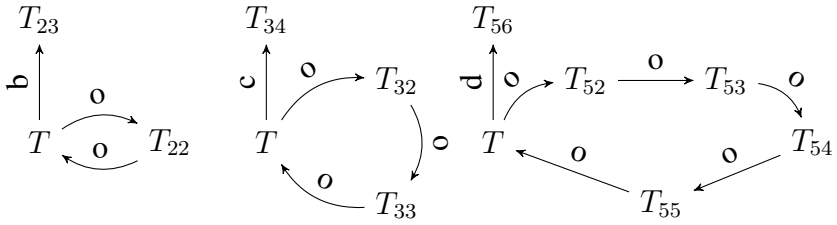


Figure 4.2: Shape Schema Graph

with the same path in U_0 as m in U, n_1 is bisimilar to m and m is bisimilar to n_2. By transitivity of bisimulation, we get that $n_1 \nleftrightarrow n_2$, a contradiction. □

Now, we show the complexity of the construction of U_0.

Theorem 4.3.8. *We can construct a size-minimal universal simulation solution U_0 in time polynomial in the size of I and by exponential in the size of* **S**. *The bound of U_0 is tight.*

Proof. Let J_0 be the core pre-solution to I w.r.t. $\mathscr{E}$ and $G_{\mathscr{E}}(I)$ the completed graph of I w.r.t. $\mathscr{E}$. We construct a typed graph as follows $U_0 = J_0 \cup G_{\mathscr{E}}(I)/_{\leftrightarrow}$ where $G_{\mathscr{E}}(I)/_{\leftrightarrow}$ is the bisimulation quotient of $G_{\mathscr{E}}(I)$. By Lemma 4.3.4 and Lemma 4.3.3, U_0 is the minimal universal simulation solution to I w.r.t. $\mathscr{E}$.

Finally, we show that the bound of U_0 is tight.

Claim 4.3.4.1. For any $m \in \mathbb{N}$, there is a constructive data exchange setting $\mathscr{E}$ such that the size of $\mathscr{E}$ is linear in m and the size of U_0 is exponential in m.

Proof. Take $m \in \mathbb{N}$. Let $I = \{R(1)\}$, Σ_{st} contains only $R(x) \Rightarrow T(f(x))$ and **S** a shapes schema (shown in Figure 4.2) that contain cycles of length $2, 3, 5, \ldots$ prime numbers with one shape name different such as T_{23}, T_{34}, and T_{56}. Let P_m stands for the m-th prime number. When constructing the minimal universal simulation solution U_0 we can observe that $|U_0| \equiv 1 (\mathrm{mod}\ 2)$ and $|U_0| \equiv 1 (\mathrm{mod}\ 3)$ and so on. Then, we can apply the Chinese reminder theorem such that $|U_0| \equiv 1 (\mathrm{mod}\ k)$ 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 minimal universal simulation solution U_0 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

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

The size of $\mathscr{E}$ is the sum of the size of the relational schema and the size of the shapes schema and the size of Σ_{st}. Since the size of $\mathbf{R}$ and Σ_{st} are 1, then the size of $\mathscr{E}$ is linear in m where m is the number of shape names. Finally, we get for $m \in \mathbb{N}$, the size of $|U_0|$ is asymptotic to $2^{2m/3}$. This ends the proof of Theorem 4.3.8. □

□

Computing certain answers

Based on the property of simulation defined previously, certain answers can be computed for Boolean queries that are robust under simulation. Thus, we state our theorem as follows.

Theorem 4.3.9. *Let $\mathscr{Q}$ be a class of Boolean graph queries robust under simulation. For any query $Q \in \mathscr{Q}$ and for any universal simulation solution U to I w.r.t. $\mathscr{E}$, $true$ is the certain answer to Q w.r.t. I and $\mathscr{E}$ if and only if $true$ is the answer to Q in U.*

Proof. Fix a family of Boolean queries $\mathscr{Q}$ that are robust under simulation. Take any query $Q \in \mathscr{Q}$. Take a universal simulation solution U to I w.r.t. $\mathscr{E}$.

For the $\Rightarrow$ direction. We assume $true$ is certain answer to Q w.r.t. I and $\mathscr{E}$ i.e., $true$ in all solutions J to I w.r.t. $\mathscr{E}$. Take a solution J. By Lemma 4.3.2, U is simulated by the solution J. By property of queries that robust under simulation, we conclude $true$ is an answer to Q in U.

For the $\Leftarrow$ direction. It suffices to notice that a universal simulation solution is also a solution as shown in Lemma 3.3.4. □

Data complexity

Now, we evaluate the tractability of our framework for the problem of computing certain answers to a query w.r.t. I and $\mathscr{E}$. This evaluation is done using data complexity. Data complexity of evaluating NREs is known to be PTIME [Pérez *et al.* 2010]. Thus, the evaluation of our approach with forward NREs is stated in the following theorem.

Theorem 4.3.10. *The data complexity of computing certain answers to forward NRE-based queries w.r.t. $\mathscr{E}$ is in PTIME.*

Proof. Take a forward NRE-based query E and an instance I of $\mathbf{R}$. It is known in [Pérez *et al.* 2010] that the evaluation of a query E in a graph G is polynomial in the size of the graph i.e., $O(|G| \cdot |E|)$. Here the forward NRE E is evaluated in a minimal universal simulation solution U_0 to I w.r.t. $\mathscr{E}$. By proof of Theorem 4.3.8, the size of $\mathscr{E}$ is linear in the size of $\mathbf{S}$ and constructing U_0 is tight. Since $\mathbf{S}$ is fixed, then deciding to evaluate a forward NRE in U_0 is in PTIME. □

4.4 Conclusion

The current chapter shows an adaptation of existing results in relational data exchange to certain query answering in relational to RDF data exchange. Our approach proposes a family of Boolean graph queries that are robust under simulation where computing certain query answering is decidable. We have shown that forward NRE-based query language is robust under simulation. We have introduced the notion of universal simulation solution, which can be constructed for any constructive data exchange setting, that allows to easily compute certain answers to any query class robust under simulation.

We have achieved attractive complexity for certain query answering. Our approach is different from the adaptation of existing results, as seen in Table 4.1 where M. is the acronym for materialization. We treat a family of queries that are incomparable with the family of queries treated in the existing results. Moreover, there is no restriction in the shapes schemas, so we cover any kind of data exchange settings.

It remains an open question to explore full NRE-based query language and investigate if computing certain query answering is decidable. We envisage as a possible idea

Approach	$\mathcal{Q}$	$\mathcal{F}_{\mathcal{E}}$	M.	Complexity
Guarded tgds	BCQ	Any	No	PTIME
Super-weakly acyclic tgds	CQ	Weakly-recursive	Yes	PTIME
Simulation	forward NRE	Any	Yes	PTIME

Table 4.1: Summary of contributions.

to use two way alternating tree automata (2ATA) [Vardi 1998] for infinite trees corresponding to unraveling the universal simulation solution. This idea has been applied to the closely related problem of computing certain answers to variants of regular path queries in the presence of ontologies [Jung *et al.* 2018, Calvanese *et al.* 2014].

4.5 Related work

A considerable amount of work has been done in the area of certain query answering [Fagin *et al.* 2005a, David *et al.* 2010, Arenas & Libkin 2008, Arenas *et al.* 2013, Pérez *et al.* 2010, Calvanese *et al.* 2014]. We present those that are close to our work and compare them.

Query answering for relational data exchange. Fagin et al. [Fagin *et al.* 2005a] use a universal solution to compute certain answers to conjunctive queries. The difference with our contribution is that we compute certain answers to forward NREs that is incomporable to conjunctive queries and we compute certain answers for any family of constructive data exchange settings while their approach can be adapted to a fragment of constructive setting where the shapes schema has no cycles in its shape graph or every cycle is only composed of weak edges.

Marnette [Marnette 2009] introduces the class of super-weakly acyclic tgds that generalizes the class of weakly acyclic tgds identified by Fagin et al. [Fagin *et al.* 2005a] that has the same properties with respect query answering. We have used their results in Section 4.2.1. The same difference in the family of queries as done with Fagin et al., and the result of Marnette is only applicable to constructive settings where shapes schemas are weakly-recursive while our approach is applicable to any constructive setting.

Several other results exist on certain query answering, which use chase termination [Calì *et al.* 2012b, Baget *et al.* 2011], or some finite representation of an infinite universal solution. Those works identify properties of family of tgds that allows to ob-

tain chase termination. In particular, Calì et al. [Calì *et al.* 2012a] present a finite structure based on the chase of guarded tgds. We have used their results in Section 4.2.2. Since the shapes schema is captured by a set of tgds that is guarded, then we do not detail the other classes. The difference with our contribution is that we materialize a solution.

Arenas et al. [Arenas *et al.* 2013] propose the use of query rewriting to compute certain answers in a data exchange context. In this sense, given a data exchange setting $\mathscr{E}$, given an instance I of $\mathbf{R}$, a query is rewritable under the $\mathscr{E}$ if there is a first order formula over the target schema such that a universal solution to I w.r.t. $\mathscr{E}$ satisfies this formula. But it is undecidable whether a query is rewritable over a universal solution. Consequently, Arenas et al. identify conditions to know if a query admits a rewriting over a universal solution. We can adapt this technique to our context but since it uses a universal solution, then their approach will only work for weakly-recursive shapes schemas. It is known that if a query is rewritable then query answering is AC0 in data complexity, which will be better for the particular class of weakly-recursive shapes schemas.

Query answering for XML data exchange. David et al. [David *et al.* 2010] study how to compute certain answers for XML queries. Authors define the notion of maximal description of a set of databases as a finite set of formulas over the query language that is satisfied in all structures of databases. In XML, the query language is defined by tree patterns, which is related to forward NREs because forward NRE does not use the inverse operator and NRE and queries descendant nodes as tree pattern queries children nodes. Thus, for a set of XML trees, a pattern is a maximal description if it is modeled in all XML trees. This pattern is called by authors a certain answer, because there exist a homomorphism from the pattern to every tree. Authors apply their result to the XML data exchange context obtaining PTIME in data complexity. The contributions of this work cannot be used in our context because there is no materialization of a good solution and tree patterns work only for trees. However, the notion of maximal description is similar to the universal simulation solution because every solution is simulated by a universal simulation solution.

Query answering for RDF. The majority of queries for RDF are based on graph pat-

terns [Barceló *et al.* 2011]. The standard language for querying RDF is SPARQL that is based on graph pattern-matching expression. In particular, Pérez et al. [Pérez *et al.* 2010] present an extension of SPARQL using NREs to navigate RDF data. We use a fragment of NREs to define a family of queries where we ensure that any query of this family on a graph has certain answers. Nikolaou and Koubarakis [Nikolaou & Koubarakis 2016] introduce the notion of representation system to answer SPARQL queries over a set of incomplete RDF databases. They identify a fragment of SPARQL where certain answers are ensured by querying the representation system. This notion of representation system is not a solution and the data complexity, which is coNP-complete, is higher than ours.

Ciucanu [Ciucanu 2015] studies the problem of query answering in the context of relational to graph data exchange. He considers only egds as target constraints and defines a family of queries using full NREs. He shows that deciding computing certain answers to full NRE-based queries is coNP-hard. His work does not subsume our contribution because the set of dependencies that captures shapes schema includes not only egds but also tgds.

Another language based on graph patterns used for querying RDF is *regular path queries* (RPQs) [Cruz *et al.* 1987]. Barceló et al. [Barceló *et al.* 2011] study the problem of certain query answering on graphs, but not in the context of data exchange, which means that no solution is materialized. Also, authors study many family of queries and show that the data complexity of computing certain answers to a query in those families ranges from NLOGSPACE to coNP. It remains an open question to adapt their work to our context.

Query answering for knowledge bases. The following works are not in the context of data exchange but they treat the problem of certain query answering on knowledge bases. Calvanese et al. [Calvanese *et al.* 2014] study certain query answering with positive 2-way regular path queries (P2RPQS). They take account of constraints in the knowledge base as we consider target constraints in the target schema. The drawback with their approach is that the data complexity of computing certain answers, which is coNP-complete, is higher than classical approaches and also from our approach. Bienvenu et al. [Bienvenu *et al.* 2013] study the complexity of answering queries over

knowledge bases. Authors consider conjunctive two-way regular path queries (C2RPQ) and the data complexity is NL-complete. Since their approach is not in the data exchange context, then there is no materialization of solution. Also, they do not consider target constraints.

Chapter 5

Visual mapping language

5.1 Motivation and use case

The majority of relational, XML and graph data exchange frameworks [Fagin *et al.* 2005a, Arenas & Libkin 2008, Boneva *et al.* 2015] use a logical language to express the rules for exchanging data. There are implementations [Marnette *et al.* 2011, Fagin *et al.* 2009, Raffio *et al.* 2008] of those frameworks that define a *visual language* for expressing the rules to reduce the difficulties of learning a logical language that a user can have. Authors of these implementations develop visual languages on top of *text-based languages*, such as SQL.

In the relational to RDF data exchange, an example of text-based language is R2RML. Sicilia et al. [Sicilia *et al.* 2017] and Crotti et al. [Junior *et al.* 2017] have defined visual languages for R2RML in their tools. Such visual languages facilitate the specification of mappings. However, we cannot use these tools in the data exchange from relational database to RDF in presence of shapes schema. In the following use case, we illustrate how our tool helps a database administrator to define mappings.

Example 5.1.1. Consider the database of software bug reports, presented in Figure 5.1. 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. We wish to export the contents of the above relational database to RDF for using by an existing application. The RDF will use three distinct IRI prefixes the default prefix `ex:` for the predicates, `bug:` for bugs, and `usr:` for users. The prefix `ex:` stands for https://example.com/, the prefix `bug:` stands for

https://example.com/ShBug and `usr:` stands for https://example.com/ShUser.

User

uid	*name*
1	Jose
2	Edith

Email

uid	*mail*
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 5.1: Relational source database

The application expects the RDF document to adhere to a ShEx schema. Suppose that this ShEx schema is expressed with shapes constraints:

$$\mathsf{ShBug} \rightarrow \mathsf{ex{:}descr} :: Lit^{1};\ \mathsf{ex{:}rep} :: \mathsf{ShUser}^{1};\ \mathsf{ex{:}related} :: \mathsf{ShBug}^{*}$$

$$\mathsf{ShUser} \rightarrow \mathsf{ex{:}name} :: Lit^{1};\ \mathsf{ex{:}email} :: Lit^{1};\ \mathsf{ex{:}tracks} :: \mathsf{ShBug}^{+}$$

This schema defines two types of nodes: ShBug for bug reports and ShUser for user info. This shapes schema happens to closely mimic the structure of the relational database with two exceptions: the type ShUser 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 *Bu2iri* and *Us2iri* that concatenate a value of the argument with the prefix `bug:` and `usr:` respectively.

Given the database instance, the shapes schema and the set of IRI constructors presented below, a database administrator wants to export this database instance to an RDF graph that satisfies the shapes schema in some practical way. But, the administrator is not able to write mappings. Instead, the administrator can load the definition of the relational schema with DDL statements and the source instance of this relational schema from a SQL script and load the shape definition from JSON script in our tool. Then, the database administrator will see boxes with text that either represent a table with its attributes or a shape name with its triple constraints. Then graphically, the database administrator defines mappings by drawing *arrows* from boxes of the relational schema to boxes of the shapes schema as seen in Figure 5.2. This set of arrows seen in Figure 5.2 corresponds to the following mappings:

$$
\begin{aligned}
&\mathit{Bug}(x,y,z) \wedge \mathit{Rel}(x,w) \Rightarrow && \mathsf{ShBug}(\mathit{Bu2iri}(x)) \wedge \\
& && \mathit{Triple}(\mathit{Bu2iri}(x), \mathsf{ex{:}descr}, y) \wedge \\
& && \mathit{Triple}(\mathit{Bu2iri}(x), \mathsf{ex{:}rep}, \mathit{Us2iri}(z)) \\
& && \mathit{Triple}(\mathit{Bu2iri}(x), \mathsf{ex{:}related}, \mathit{Bu2iri}(w)) \\
&\mathit{User}(x,y) \wedge \mathit{Email}(x,z) \wedge \mathit{Track}(x,w) \Rightarrow && \mathsf{ShUser}(\mathit{Us2iri}(x)) \wedge \\
& && \mathit{Triple}(\mathit{Us2iri}(x), \mathsf{ex{:}name}, y) \wedge \\
& && \mathit{Triple}(\mathit{Us2iri}(x), \mathsf{foaf{:}mbox}, z) \wedge \\
& && \mathit{Triple}(\mathit{Us2iri}(x), \mathsf{ex{:}tracks}, \mathit{Bu2iri}(w))
\end{aligned}
$$

Then, the user obtains a solution in Turtle format as seen in Figure 5.3. In this solution, we observe that the symbol "@@@" used in an IRI denotes fresh IRI nodes and alone itself denotes null literals. We discuss later on the use of symbol "@@@". □

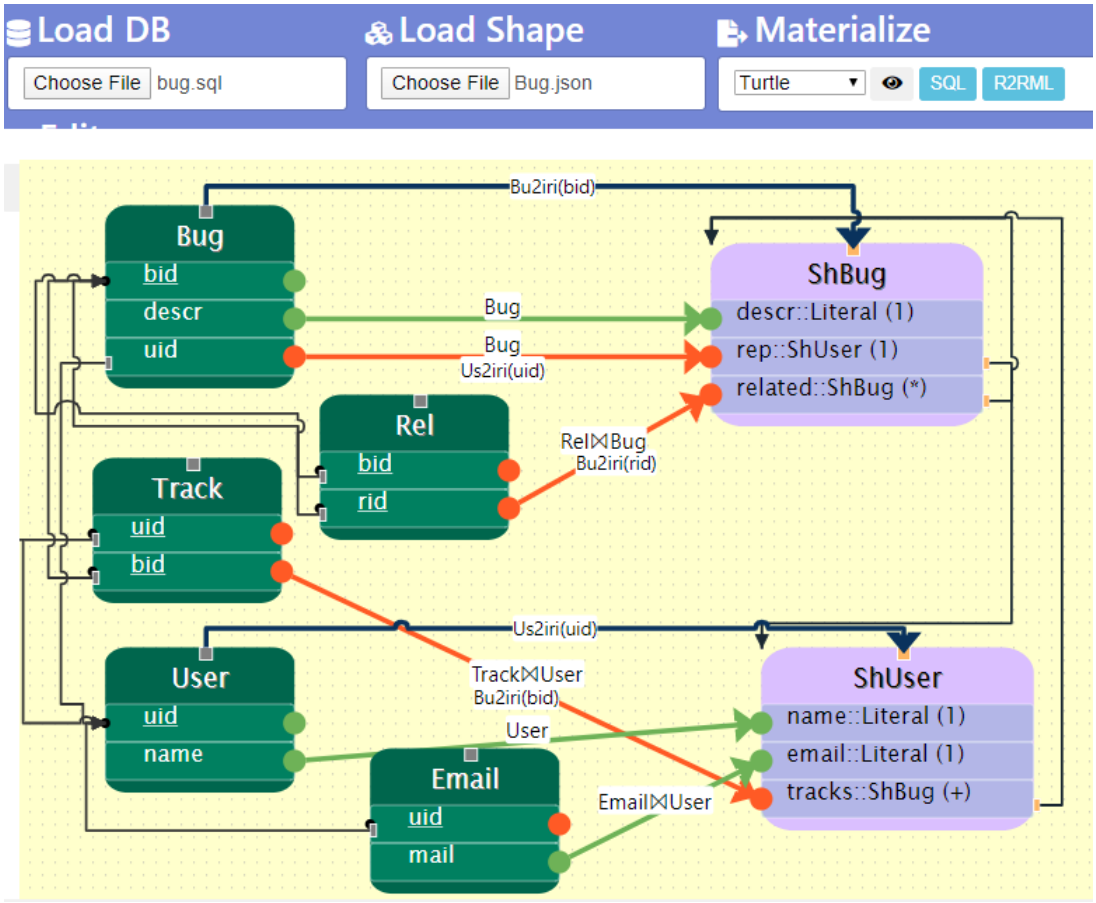


Figure 5.2: ShERML.

```
<https://example.com/ShBug/1>
  <https://example.com/descr> "Boom!";
  <https://example.com/rep> <https://example.com/ShUser/1>.
<https://example.com/ShBug/2>
  <https://example.com/descr> "Kabang";
  <https://example.com/rep> <https://example.com/ShUser/1>.
<https://example.com/ShBug/3>
  <https://example.com/descr> "Bang!";
  <https://example.com/rep> <https://example.com/ShUser/2>.
<https://example.com/ShUser/1>
  <https://example.com/name> "Jose";
  <https://example.com/email> "j@ex.com";
  <https://example.com/tracks> <https://example.com/ShBug/2> ,
                    <https://example.com/ShBug/1>.
<https://example.com/ShUser/2>
  <https://example.com/name> "Edith";
  <https://example.com/email> "@@@";
  <https://example.com/tracks> <https://example.com/ShBug/@@@>.
<https://example.com/ShBug/@@@>
  <https://example.com/descr> "@@@";
  <https://example.com/rep> <https://example.com/ShUser/@@@>.
<https://example.com/ShUser/@@@>
  <https://example.com/name> "@@@";
  <https://example.com/email> "@@@";
  <https://example.com/tracks> <https://example.com/ShBug/@@@>.
```

Figure 5.3: Solution in Turtle format.

5.2 Preliminary notions

In this section, we recall basic notions of data visualization and human-computer interaction used to develop ShERML.

Visual representations. Interaction between users and computers is done by a graphical interface that uses *visual representations* of data. Visual representations [Lohse *et al.* 1994] are representations assigned to data that use visual variables (graphical objects in screen) to convey the information to the user. A visual object can be a picture, a geometric form, or text with color or composition of them.

Visual representations can be more effective than symbolic information such as text [Treisman 1985] because they amplify the cognition by providing the human more information, faster and with less cognitive effort. Thus, for non-expert users, the learning of text-based languages requires a considerable amount of time.

Visual language. Visually encoding on top of a text-based language is one way to en-

able non-expert users the usage of text-based languages. As a result, we have a *visual language*, which is a language that uses visual representations to depict words of a text-based language $\mathscr{A}$. The alphabet of a visual language is a set of visual objects and visual representations.

Interaction paradigms. The purpose of an interaction model is to provide guidelines for guiding designers and developers to create interactive systems. One such interaction model is *direct manipulation* [Shneiderman 1983]. This paradigm is based on *visual representations* and establishes a guideline when designing an interactive tool. This guideline consists of four principles: choosing a visual representation, learning components of the visual representation, setting operations with the visual representation and displaying visual representations on the screen. The first principle provides techniques to clearly and accurately represent the information. The second principle provides techniques to measure the complexity of learning the visual representation by a user. The third principle provides interaction techniques such as *drag and drop*. This interaction technique consists of enabling the user to move visual objects, change a sorting order or put a visual object in a particular place. The last principle provides techniques for arranging visual representations so the information is clear to the user. Hutchins et al. [Hutchins *et al.* 1985] show that tools following these principles facilitate the tasks of users.

5.3 The intermediary language

In this section, because VML is constructed on top a text-based language, we present an intermediary language called R2VML. It is a declarative text-based representation of the mapping language based on logical rules that uses vocabulary of SQL that is familiar for users with basic background on databases. R2VML mappings capture a fragment of constructive st-tgds specified in the data exchange framework presented in this manuscript as seen in Figure 5.4.

We can observe that writing R2VML mapping is more intuitive than writing st-tgds for a user that has background on databases. R2VML is presented without using variables compared to st-tgds and with a Turtle-like style, which is familiar for RDF users. Also, R2VML constraints the way of writing mappings such that our tool ShERML

```
Bug join Rel
=>
Bu2iri(Bug.bid) as ShBug
 ex:descr Bug.descr;
 ex:rep Us2iri(Bug.uid);
 ex:related Bu2iri(Bug.rid).
```

$$\begin{aligned}&\textit{Bug}(x,y,z) \wedge \textit{Rel}(x,u) \Rightarrow \\ &\quad \mathsf{ShBug}(\textit{Bu2iri}(x)) \wedge \\ &\quad \textit{Triple}(\textit{Bu2iri}(x), \mathsf{ex{:}descr}, y) \wedge \\ &\quad \textit{Triple}(\textit{Bu2iri}(x), \mathsf{ex{:}rep}, \textit{Us2iri}(z)) \wedge \\ &\quad \textit{Triple}(\textit{Bu2iri}(x), \mathsf{ex{:}related}, \textit{Bu2iri}(u))\end{aligned}$$

```
User join Email join Track
=>
Us2iri(User.uid) as ShUser
 ex:name User.name;
 ex:email Email.mail;
 ex:tracks Bu2iri(User.bid).
```

$$\begin{aligned}&\textit{User}(x,y) \wedge \textit{Email}(x,z) \wedge \textit{Track}(x,w) \Rightarrow \\ &\quad \mathsf{ShUser}(\textit{Us2iri}(x)) \wedge \\ &\quad \textit{Triple}(\textit{Us2iri}(x), \mathsf{ex{:}name}, y) \wedge \\ &\quad \textit{Triple}(\textit{Us2iri}(x), \mathsf{foaf{:}mbox}, z) \wedge \\ &\quad \textit{Triple}(\textit{Us2iri}(x), \mathsf{ex{:}tracks}, \textit{Bu2iri}(w))\end{aligned}$$

Figure 5.4: Comparison example of R2VML mappings and st-tgds.

is capable of processing. Another reason for having R2VML is that in general some users prefer to have graphical and text-based mappings. Moreover, users want to understand the model behind the VML mappings, and so R2VML presents a user-friendly and succinct syntax for defining mappings.

Now, given a library of IRI constructors $\mathbf{F} = (\mathscr{F}, F)$, the relational schema $\mathbf{R} = (\mathscr{R}, \mathit{attrs}, \Sigma_{\text{fd}}, \Sigma_{\text{ind}})$, the shapes schema $\mathbf{S} = (\mathscr{T}, \delta)$, we define a R2VML mapping with the following the grammar:

RV ::= $\mathit{RelExpr}$ "=>" $\mathit{TriExpr}$ | $\mathit{RelExpr}$ "`where`" CExpr "=>" $\mathit{TriExpr}$

$\mathit{RelExpr}$ ::= R | $\mathit{RelExpr}$ "`join`" $\mathit{RelExpr}$

CExpr ::= R"." Aname Op Nro | R"." Aname Op Str

$\mathit{TriExpr}$ ::= F"(" R"." Aname")" "`as`" T $\mathit{PropExpr}$

$\mathit{PropExpr}$ ::= p Obj";" $\mathit{PropExpr}$ | p Obj"."

Obj ::= F"(" R"." Aname')" | R"." Aname | Fun"(" R"." Aname")"

Op ::= "<" | "<=" | ">" | ">=" | "=" | "`like`"

AName ::= name of attribute in a relation

Fun ::= trim | upper | lower

Nro ::= a number

Str ::= a string value

where $R \in \mathscr{R}$, $F \in \mathscr{F}$, $T \in \mathscr{T}$, $p \in \mathsf{Prop}_{\mathbf{S}}$.

A rule (RV) is composed of a relational expression ($RelExpr$) and a triple expression ($TriExpr$). Both expressions are separated by the symbol =>. Also, a rule can contain a conditional expression ($CExpr$) separated from the relational expression by the string `where`. The relational expression is either a relational name R or a set of relation names combined by the string `join`. The conditional expression is specified by a relation name, an attribute name ($Aname$), an operator (Op) and a parameter either number (Nro) or string (Str). The triple expression is composed of an IRI constructor (F), the type name (T) and the property expression ($PropExpr$). The argument of the IRI constructor is inside a parenthesis and is composed of a relation name and an attribute name separated by the symbol dot. The property expression is composed of a property (p) and an object (Obj), which is either an IRI constructor or a relational name with an attribute name or a string function Fun such as $trim, upper$ and $lower$ that converts the value of an attribute name.

The interpretation of the grammar is as follows. The identifier $trim$ is interpreted as the function that removes all spaces from text except for single spaces between words. The identifier $upper$ as the function that converts a string to uppercase letters and the identifier $lower$ as the function that converts a string to lowercase letters. The identifier R is interpreted as the name of a table in a database. The identifier F is interpreted as a function that generates an IRI using the value of the parameter. The relational expression is interpreted as a SQL query. If the conditional expression is in the rule is interpreted as the filters on the query result. The triple expression is interpreted as the generation of triples where the subject is an IRI resulting from the application of an IRI constructor with the value of the attribute. The property expression is interpreted as the set of property objects associated with the subject. We allow trim, upper and lower functions to be applied to data values of the object. The triple objects come from the values of the attributes specified in the property expression. Some objects can be IRIs resulting from the application of an IRI constructor with the value of the attribute.

The triple expression is expressed as a Turtle syntax that is a natural way to declare RDF graphs except the specification of types. The subject of the triple expression includes in the grammar the symbols "`as`" T that indicates the type of the subject nodes that will be generated. It is created a monadic fact with the name T and the subject node. This constitutes the typing relation that together with the set of triples is the typed

graph.

Now, for human-computer interaction reasons, we restrict the kind of R2VML mappings that can be defined in ShERML. We set a restriction over the set of tables, which we call a *path*, used in a query of R2VML. In ShERML, the drawing of a green or orange arrow requires a path to be specified. We propose a set of paths, so a user is not concerned to write it. A path in this set has the property that for every two tables of the path, there exists a foreign key such one table references the other. This property guarantees that a st-tgd is uniquely defined. For instance, suppose we have the following two relations *User*($\underline{uid}, name$) and *Track*($\underline{tid}, uid, track{-}nbr$). Also, we have the following R2VML.

```
User join Track => Us2iri(User.uid) as ShUser;
                   ex:name User.name;
                   ex:track Track.track-nbr.
```

This R2VML mapping can correspond to the following st-tgd

$$\textit{User}(x, y) \wedge \textit{Track}(u, x, v) \Rightarrow \textit{Triple}(\textit{Us2iri}(x), \mathsf{ex{:}name}, y) \wedge \textit{Triple}(\textit{Us2iri}(x), \mathsf{ex{:}tracks}, v) \wedge \mathsf{ShUser}(\textit{Us2iri}(x)) \quad (5.1)$$

or can correspond to the two following st-tgds

$$\textit{User}(x, y) \Rightarrow \textit{Triple}(\textit{Us2iri}(x), \mathsf{ex{:}name}, y) \wedge \mathsf{ShUser}(\textit{Us2iri}(x)), \quad (5.2)$$

$$\textit{User}(x, y) \wedge \textit{Track}(u, x, v) \Rightarrow \textit{Triple}(\textit{Us2iri}(x), \mathsf{ex{:}tracks}, v). \quad (5.3)$$

These two corresponding sets of st-tgds generate two different graphs from a given instance. A graph generated from st-tgd 5.1 will have nodes of type ShUser only if there is a user that has some track-nbr. While in a graph generated from st-tgds 5.2 and 5.3, the number of nodes of type ShUser will be as many users are stored in *User*. But, if we assume the existence of a reference from uid of *Track* to *User*, then the unique correspondence of the R2VML mapping is the st-tgd 5.1.

To define a VML mapping, green and orange arrows are grouped by the first element of the path, which is the table that is connected by a blue arrow. We describe later on how a VML mapping is created. To define the path to be specified in a R2VML mapping

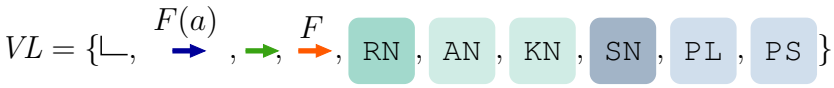
from these arrows, the paths specified on the arrows are combined in a single path such that there is no repetition of tables and the first element of the path is the table that the blue arrow connects.

To visually represent a R2VML mapping in a directly way and to define a unique st-tgd from a R2VML mapping, R2VML is well defined if the primary key of the first table of the path π is the argument for the IRI constructor and if there is a path π' composed with the tables of path π such that for every two elements of π', there is a foreign key from one to the other.

5.4 The visual mapping language

In this section, we define a visual mapping language (VML). It is composed of different types of boxes, arrows that have a label or not, and an orthogonal line that is used to relate boxes. Each element of VML has its correspondence to an element of R2VML and its semantics follows from there.

The alphabet of VML is composed of the following visual objects:



where

- F is the identifier for an IRI constructor and a is the identifier of the argument for the IRI constructor,
- `RN` is the identifier for a relational name in $\mathscr{R}$,
- `AN` is the identifier for an attribute name of a relational name,
- `KN` is the identifier for an attribute name that is the primary key in a relation name,
- `SN` is the identifier for a shape name in $\mathscr{T}$,
- `PL` is the identifier for a property name such that there is a triple constraint in the shapes schema with this property where the target type is Lit,

- PS is the identifier for a property name such that there is a non-literal type S and a triple constraint with this property name and target type S.

For instance in Figure 5.2, we observe the blue arrow with label *Us2iri*(uid) that goes from relation name *User* to shape name ShUser. Here the relational name *User* corresponds to RN, the shape name ShUser to SN and the label *Us2iri*(uid) to $F(a)$. Then, we observe the green arrow with label *Email* $\bowtie$ *User* that goes from attribute mail of *Email* to the triple constraint with property tracks :: ShBug of shape ShUser. Here the relational names *User* and *Email* corresponds to RN, the attribute $mail$ corresponds to AN, the shape ShUser corresponds to SN, email corresponds to PL, the path of *Email* $\bowtie$ *User*, the dependence of the attribute to a table and the dependence of a triple constraint to its shape corresponds to the orthogonal line.

Now, given a library of IRI constructors $\mathbf{F} = (\mathscr{F}, F)$, the relational schema $\mathbf{R} = (\mathscr{R}, attrs, \Sigma_{\text{fd}}, \Sigma_{\text{ind}})$, the shapes schema $\mathbf{S} = (\mathscr{T}, \delta)$, we define VML mappings with the following grammar:

VR ::= RN VB $\xrightarrow{F(a)}$ SN

VB ::= ⌞ RN VB | ⌞ AN → ⌞ PL | ⌞ KN $\xrightarrow{F}$ ⌞ PS

A visual expression VR allows to relate a query with a shape and its triple constraints and defines a VML mapping. A VML mapping is composed of a set of visual objects that represents the relational expression and triple expression of an R2VML mapping. The mapping starts with a blue arrow between a source box with RN and a target box with SN. This arrow has as a label an IRI constructor and specifies that the primary key of the source box is converted to an IRI and is typed with the shape name SN. The attribute name either AN or KN comes from the result query. The target box of a green arrow is specified only to property PL of the related shape that comes from a triple constraint whose target type is Lit. This arrow species that the value of AN is directly used as the object of the triple that will be generated. A target box of an orange arrow is specified only to property PS of the related shape that comes from a triple constraint whose target type is a shape. This arrow specifies that the value of attribute KN is converted to an IRI by the application of F where its interpretation is given in the library $\mathbf{F}$.

We illustrate in Figure 5.5, a R2VML mapping and its correspondent VML map-

ping. This VML mapping indicates how the graph will be populated with properties of ShUser where the object values of triples come from the attribute values. In the case of the orange arrow, there is a conversion of a value of the attribute bid with the IRI constructor *Bu2iri* that is in the label of the orange arrow. Here the subjects of triples are obtained from the application of *Us2iri* to the values of uid. This VML mapping is constructed from the R2RML mapping in Figure 5.5 as follows. The query *User* `join` *Track* is mapped to the two boxes related to the orthogonal line. The subject *Us2iri*(uid) `as` ShUser is mapped to the blue arrow with *Us2iri*(uid) as a label where the target box has the shape name ShUser. The property expression ex:name ShUser.$name$ is mapped to the green arrow where the source box has the attribute $name$ that comes from the query and the target box has the property label ex:name that comes from a triple constraint in the definition of ShUser where the target type is Lit. The property expression ex:tracks *Bu2iri*$(Track.bid)$ is mapped to the orange arrow with *Bu2iri* as a label where the source box has the attribute name bid that also comes from the query and target box has the property label ex:tracks that comes from a triple constraint in the definition of ShUser but the target type is the shape ShBug.

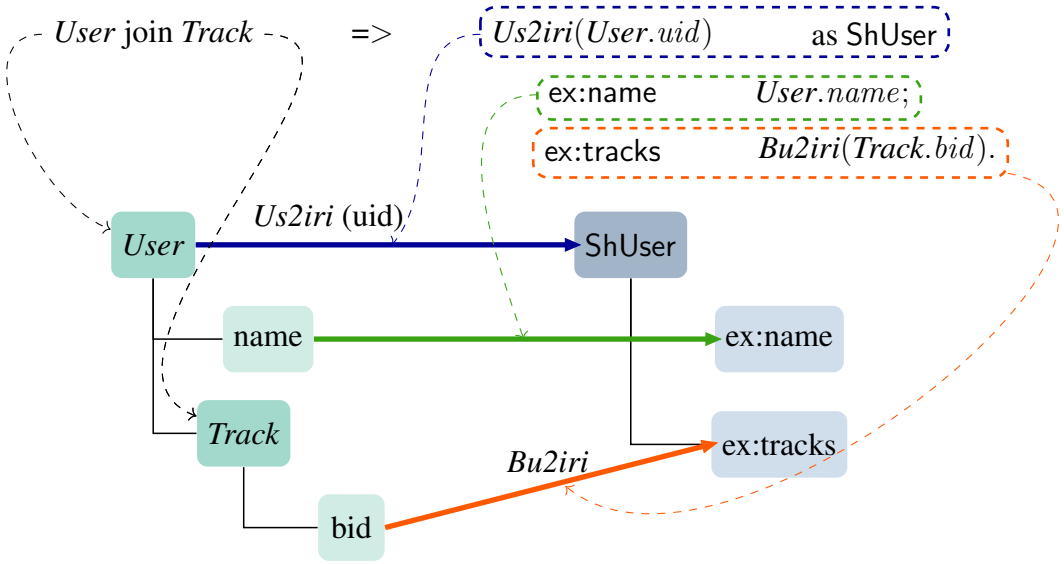


Figure 5.5: A correspondent VML mapping of a R2VML mapping.

Now, we describe in Table 5.1 the correspondence between the visual objects of a VML mapping and elements of R2VML mapping.

Based on the correspondence given in Table 5.1, we can define the semantics of

Visual object	Correspondence to
RN	R used in the relational expression and the triple expression.
SN	T used in the triple expression after the string "as".
└ RN	R in the relational expression but after the string "join".
└ AN	$Aname$ when the object in the property of the triple expression is of the form either $R.Aname$ or $Fun(R.Aname)$.
└ KN	$Aname$ but when the object in the property expression is of the form $F(R.Aname)$.
└ PL	p in the property expression when the object is of the form either $R.Aname$ or $Fun(R.Aname)$.
└ PS	p when the object in the property expression is of the form $F(R.Aname)$.
$F(a)$ →	"=>" and $F(R.A.name)$ in the triple expression that appears before the string "as".
→	"=>" and the writing of the property expression p $R.Aname$ or p $Fun(R.Aname)$.
F →	"=>" and the writing of the property expression p $F(R.Aname)$.

Table 5.1: Correspondence of grammar of VML with grammar of R2VML.

VML mappings as follows:

RN $\xrightarrow{F(a)}$ SN It defines how rows from table RN are to be mapped to IRI nodes using the IRI constructor F with the value of the primary key a satisfying the shape constraint SN. We call this mapping an *IRI mapping*.

└ AN → └ PL It defines how table rows are mapped to triples where the predicate of every triple is the value of PL and the objects are values of AN. This VML mapping depends on the existence of an IRI mapping. We call this mapping a *property mapping*.

└ KN $\xrightarrow{F}$ └ PS It defines how table rows are mapped to triples where the predicate of every triple is the value of PS and the objects are IRI nodes that are obtained from the application of F with the values of the attribute KN. As the property mapping, it depends on the existence of an IRI mapping. We call this mapping a *reference mapping*.

The other elements of the relational expression in an R2VML mapping are used as

annotations over the visual objects such as the conditional expression and string functions. Figure 5.6 shows an example of a VML mapping with annotations. The conditional expression has no proper visual representation as the other elements, but the conditional expression has its correspondence as a text annotation over an IRI mapping. In the case of string functions, they are annotated over a property mapping.

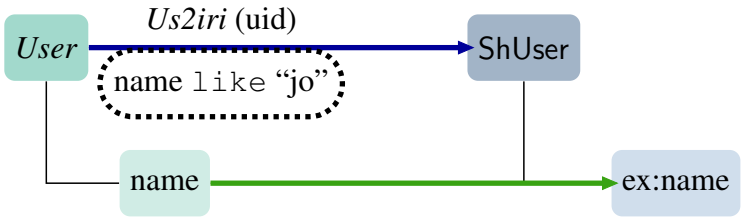
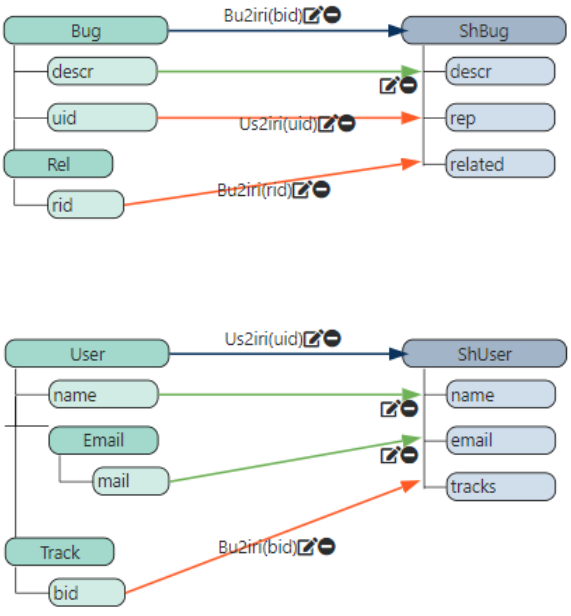


Figure 5.6: An example of VML mapping with annotations.

The language of st-tgds only captures VML mappings that do not contain a condition or string function annotations. In Figure 5.7, we observe that each VML mapping is captured by a st-tgd.



$$Bug(x, y, z) \wedge Rel(x, w) \Rightarrow \\ \mathsf{ShBug}(Bu2iri(x)) \wedge \\ Triple(Bu2iri(x), \mathsf{ex{:}descr}, y) \wedge \\ Triple(Bu2iri(x), \mathsf{ex{:}rep}, Us2iri(z)) \\ Triple(Bu2iri(x), \mathsf{ex{:}related}, Bu2iri(w))$$

$$User(x, y) \wedge Email(x, z) \wedge Track(x, w) \Rightarrow \\ \mathsf{ShUser}(Us2iri(x)) \wedge \\ Triple(Us2iri(x), \mathsf{ex{:}name}, y) \wedge \\ Triple(Us2iri(x), \mathsf{foaf{:}mbox}, z) \wedge \\ Triple(Us2iri(x), \mathsf{ex{:}tracks}, Bu2iri(w))$$

Figure 5.7: Example of VML mappings captured by st-tgds.

5.5 ShERML

In this section, we describe a tool for data exchange from relational to RDF called ShERML. Our tool enables users to create mappings between a source relational schema and a target shapes schema. Also, ShERML verifies consistency of mappings. We have developed our tool with the direct manipulation paradigm.

5.5.1 Architecture

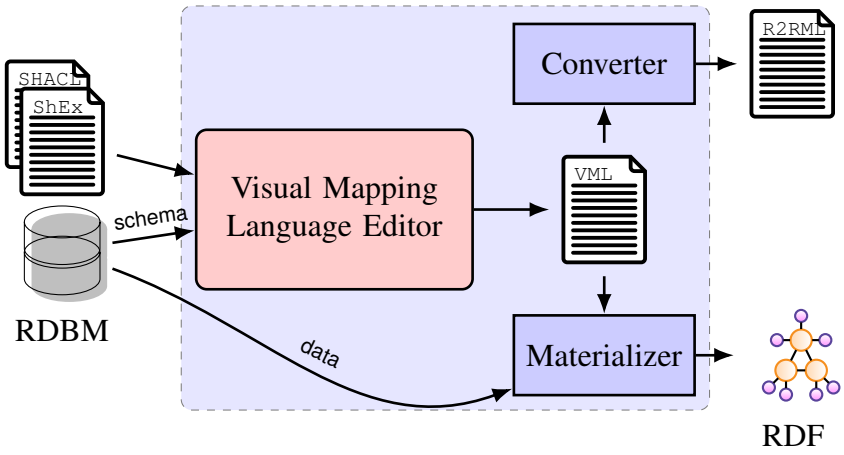


Figure 5.8: ShERML Architecture and Workflow.

We now present an overview of ShERML architecture and the flow of data in Figure 5.8. ShERML receives as input a source relational schema, an instance of the relational schema and a target shapes schema, which can be expressed with SHACL or ShEx. The output of ShERML is an RDF graph, which is the result of materializing the database instance with a set of mappings such that the target shapes schema is satisfied. ShERML has three main components: the VML editor, the materializer and the converter. The VML editor shows the visual representations of source and target schemas and allows to draw arrows. Each time an arrow is designed, a VML mapping is created. This component generates a VML script that is the input for the materializer component that executes it by running SQL queries on the relational database and constructing the desired RDF graph. Then, the converter component generates a R2RML script that can be used by third-party applications. We detail these components in the sequel.

5.5.2 VML Editor

The VML editor is the main component and is used to define mappings. The definition of mappings is done by drawing arrows with the drag and drop technique. This component receives as inputs the source and target schemas and produces a VML script. The VML editor seen in Figure 5.9 consists of two panels: the modeling and the mapping panel. The modeling panel displays graph representations for source and target schema. The mapping panel displays the VML mappings, which are created by drawing of arrows, with modify and delete buttons.

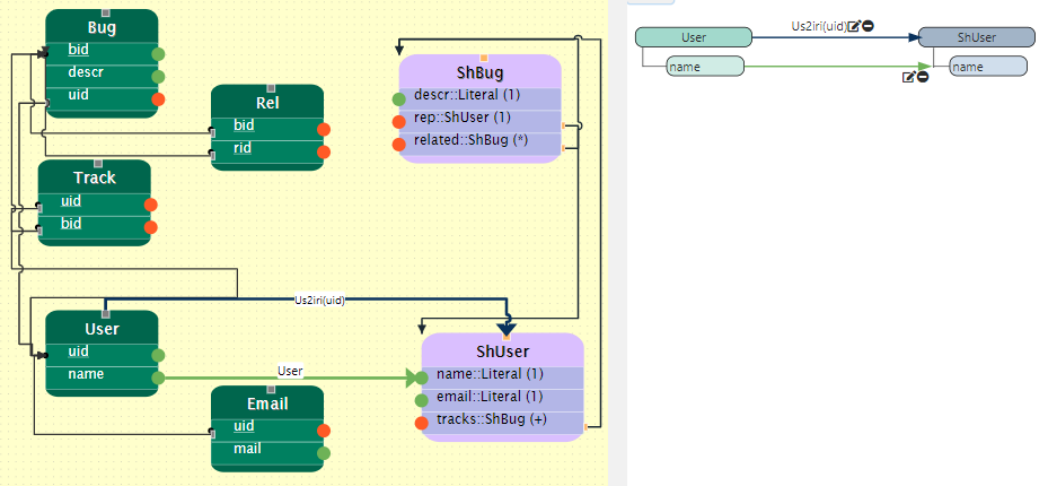


Figure 5.9: Modeling panel at left side and Mapping panel at right side of VML editor.

The input for starting the mapping process consists of loading the database and the schema for RDF which can be a ShEx or SHACL script. When the shapes schema is loaded, ShERML creates for each shape name its corresponding function symbol. This function symbol is unique and its interpretation is by default the concatenation of an URL assigned to the shape that is also unique and the value of the argument. Thus, every IRI constructor in ShERML is used with a unique shape name.

Now, we describe the implemented interactions in the editor. In the modeling panel, the user can reorder nodes and edges of the graph representations and can draw arrows from nodes of source schema to nodes of the target schema. Each arrow drawn in the VML editor creates a VML mapping displayed in the mapping panel. Each VML mapping can be modified with the modify button that allows users to add string functions on property mappings, to modify IRI constructors on an IRI and a reference mapping. Also in the modify button over an IRI mapping, a user can add filter conditions on the

attributes of the relation name. These conditions of attributes are displayed as annotations to the arrows. In the following, we describe the graph representations for source and target schema, and describe a VML script.

Graph representation for relational and shapes schema. The graph representation for the relational schema is described in Figure 5.10.

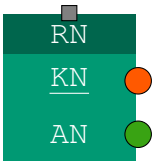
Visual object	Representation
	The whole visual object is composed of three different visual objects and represents a relation name with its attributes. The visual object for a relation name identified with RN is a box with a square anchor. The visual object for an attribute name identified with AN is composed of a box and a circular anchor of color green, which represents any attribute that is not a primary key or foreign key. The visual object identified with KN is composed of a box with the name of the attribute and a circular anchor of color orange. If this visual object has the text underline, then it represents a primary key. Otherwise, it represents a foreign key.
→	It represents an inclusion dependency. The black arrows are between relational tables.

Figure 5.10: Graph representation for relational schema.

The graph representation for the shapes schema is described in Figure 5.11. The

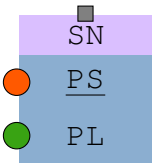
Visual object	Representation
	The whole visual object is composed of three different visual objects and represents a shape name and the set of triple constraints that the shape defines. The visual object for a shape name identified with SN is a box with a square anchor. The visual object for a triple constraint, where target type is a shape name, identified with PS is composed of a box and a circular anchor of color orange. The visual object for a triple constraint, where the target type is literal, identified with PL is composed of a box and a circular anchor of color green.
→	It represents the relation between two shape names, i.e a shape name T is related with S if there is a triple constraint with S that is the definition of T. The black arrows are between shape names.

Figure 5.11: Graph representation for shapes schema.

anchors in both graph representations are used for creating connections later on.

Visual representation of rules. We use arrows as the visual representation of rules. The arrows are from visual objects of the graph representation of the relational schema to visual objects of the graph representation of the shapes schema. There are three kinds of arrows.

IRI arrow it can only connect square anchors. It has as attribute an IRI constructor. The editor draws an IRI arrow with the name of the IRI constructor and parameter on top of the arrow. By default, the parameter is the primary key of the table. If the primary key is composed of more than one attribute, ShERML allows users to select one attribute of the primary key.

Property arrow it can only connect two green anchors. A property arrow depends on an IRI arrow. It has as attribute a path. The editor draws a property arrow with a path on top of it. If the node is connected by the property arrow has incoming or outgoing edges, ShERML proposes a set of paths where the user has to select one path. The computation of set of paths is based on the following procedure. Computes all set of paths by a graph search algorithm. Then, it is filtered only the paths where the root of the path is already connected by an IRI arrow to the target node that contains the target anchor connected by the property arrow.

Reference arrow it can only connect two orange anchors. Similarly to a property arrow, a reference arrow depends on an IRI arrow. It has as attributes an IRI constructor and a path. The editor draws the reference arrow with the IRI constructor on top of it and the path on the bottom of it. As in the property arrow, a set of paths is computed where the user selects one path. Also, the user has to select the IRI constructor.

ShERML allows users to delete arrows. If an IRI arrow is deleted then, all property and reference arrows are deleted as well.

Furthermore, in the mapping panel, the draw of a VML mapping is designed to avoid the crossing of arrows. From a blue arrow in the editor, it is analyzed all the paths contained in the set of green and orange arrows that depend on the blue arrow. First, the arrows that connect the attributes of the relation name. Then, the arrows are organized according to the length of the set of paths and if the paths have the same length

then the paths are organized in lexicographical order. Finally, arrows that connect the attributes of the same table in the path, first are created the property mappings and then the reference mappings.

VML script. The output of the editor is a VML script that contains the set of VML mappings, R2VML mappings, the graph representations of schemas and the set of positions.

5.5.3 Materializer

We recall that the problem of relational to RDF data exchange is to find a solution i.e., a graph that satisfies the shapes schema and the set of mappings. In this sense, the materializer generates an RDF graph that is a solution to an instance given as input in ShERML w.r.t. the set of mappings defined in the editor. This component receives as input a VML script and uses the R2VML mappings that are contained in it and outputs an RDF graph. In order to extract the data from the tables, we translate the R2VML mappings to SQL views that will represent the RDF data. The translation is done as follows. The materializer converts each R2VML mapping to a st-tgd. Every st-tgd is normalized and translated to a SQL query following the method used by Benedikt et al. [Benedikt *et al.* 2017]. This SQL view selects the head of a st-tgd from the data contained in relations that appear on the left side of the st-tgd. There will be two types of views: the *triple view* and the *type view*. The following SQL query illustrates the corresponding equivalence of a st-tgd.

$$Bug(x, y, z) \wedge Rel(x, z) \Rightarrow \mathsf{ShBug}(Bu2iri(x))$$

$$Bug(x, y, z) \wedge Rel(x, z) \Rightarrow Triple(Bu2iri(x), \mathsf{ex{:}rep}, y)$$

```
CREATE VIEW TBug (term,type) AS
SELECT CONCAT('bug:',bid),
'ShBug'
FROM Bug inner join Rel ON
Bug.bid=Rel.bid;
CREATE VIEW Triple AS
SELECT CONCAT('bug:',Bug.bid) as s,
'ex:name' as p,
CONCAT('usr:',Bug.uid) as o
FROM Bug inner join Rel ON
Bug.bid=Rel.bid;
```

The execution of views previously defined will generate an RDF graph, but there

can be constraints that are not yet satisfied. So, the materializer creates the relation Shape(source, prop,target,mult) in the database that stores the shape names and its triple constraints with multiplicity 1 or +, i.e., we store the obligatory shape graph (cf. Section 3.3.5). Then, the materializer constructs two views called Triples that selects all triple views and Types that select all type views. These views are used to create another view called TripleSim that generates triples with fresh nodes to satisfy the triple constraints that are not yet satisfied. This process of generating triples with fresh nodes can be infinite. We use the symbol "@@@" as the value for generating the fresh nodes so the process of generating triples arrives at fixed point. The following SQL query illustrates that fresh nodes are created (line 6) if the constraints are not yet satisfied (line 11).

```
1  CREATE VIEW TripleSim (s,p,o) AS
2    SELECT * FROM Triples
3     UNION
4    SELECT Ts.term,
5           Sh.label,
6           CASE WHEN Sh.target ='Lit' THEN '@@@'
7            ELSE CONCAT('ex:',Sh.target,'/','@@@')
8           END AS typeO
9    FROM Shape AS Sh,Types AS Ts
10   WHERE Ts.type=Sh.typeS AND
11         CONCAT(Ts.term,Sh.source,Sh.prop) NOT IN
12         (SELECT CONCAT(T.s,Ty.type,T.p)
13           FROM Triples as T, Types AS Ty
14           WHERE T.s=Ty.term);
```

Finally, the converter constructs the view Solution that selects all triples and the view TypeSol that selects all the content of the view Types and the views that typed the unknown values. The view Solution is materialized as a table with attributes sub, pred and obj. The data that will contain this table is translated into an RDF graph by using the methods provided by Apache Jena[1]. The content of the view TypeSol with the RDF graph obtained is a solution to an instance of the relational schema w.r.t. a data exchange setting. ShERML offers three formats to export the solution: Turtle, N-Triples and RDF/JSON.

[1] https://jena.apache.org/

Now, we discuss why we use the symbol "@@@" to represent the null values. To obtain a solution to a given instance w.r.t. the setting, we use the chase procedure with the st-tgds and the set of dependencies that captures the shapes schema. Since the chase can be infinite if there are cycles in the obligatory shape graph, then inventing new values each time a **PE** rule is triggered is not a good solution because the chase will never terminate and we want a finite solution. Thus, we invent a fixed IRI value for each type so a **PE** rule with the same type will be triggered only once and a fixed literal value. Because this fixed literal value must represent a null literal value we decide to use "@@@" symbol that has no meaning for a user. However, this is also a problem because the solution will have meaningless information. Here no good solution for the problem of representing null literal value exists. We use also the "@@@" symbol in the creation of a fixed IRI value for each type. Since, the chase is implemented in SQL, we cannot use blank nodes. Because of the approach adopted here, we do not generate a universal solution. For instance, the solution in Figure 5.3 for the setting and instance shown in Example 5.1.1 is not universal.

5.5.4 Converter

The converter generates an R2RML script that can be used into other applications. This component receives as input a VML script and outputs R2RML mappings. The converter uses the set of R2VML mappings contained in the script. The process of converting from R2VML to R2RML is as follows. Each rule in R2VML is converted to a triple map in R2RML. The relational expression in R2VML is inside $\mathrm{rr{:}logicalTable}$. The triple expression is translated to $\mathrm{rr{:}subjectMap}$ and $\mathrm{rr{:}predicateObjectMap}$. The IRI constructor is mapped to the IRI value of concatenation specified in the program. The expression "as" T is translated as an additional rule where the predicate map is $\mathrm{rr{:}class}$ and the object is T. The application of the predicate map $\mathrm{rr{:}class}$ will generate a triple with the predicate label rdf:type. For instance, Figure 5.12 shows the translation of R2VML mapping to R2RML mapping.

R2VML

```
User =>
  Us2iri(User.uid) as ShUser
    ex:name User.name.
```

R2RML

```
<#TriplesMap1>
  rr:logicalTable [ rr:sqlQuery
  """ SELECT uid,User.name
      FROM User""" ];
  rr:subjectMap  [
    rr:template
      "https://example.com/ShUser/{uid}";
    rr:class ShUser];
  rr:predicateObjectMap [
    rr:predicate ex:name;
    rr:objectMap
      [rr:column  "name"];].
```

Figure 5.12: An example of R2VML and R2RML mapping.

5.5.5 Consistency checking

A novelty of our tool that distinguishes from other mapping tools is the consistency verification. This functionality allows users to know if the set of VML mappings that have specified in the tool are consistent i.e., for any instance of the relational schema there is a solution. To deciding consistency we use two algorithms presented in Section 3.2.1 and 3.3.5. The set of st-tgds that captures R2VML mappings specified in ShERML belongs to a simpler class where every st-tgd has a triple and a monadic atom that types the subject term. Given a database, the application of this set of st-tgds will avoid:

- to have conflicting types in any extended graph of the core pre-solution to the source instance; and
- to compute the whole set of violation sorts and reduce the steps for constructing the set of contentious-based instances in the algorithm of Section 3.2.1.

Therefore, deciding consistency in ShERML is a simpler version of Algorithm 1.

ShERML translates the set of VML mappings to R2VML mappings and from R2VML mappings to set of st-tgds as follows. For each R2VML mapping, we take the properties associated to the subject T and IRI constructor f in the R2VML mapping and for each property p we create a rule of the form $\varphi(\mathbf{x}) \Rightarrow T(f(\mathbf{y})) \wedge \mathit{Triple}(f(\mathbf{y}), p, t)$ for $\mathbf{y} \subseteq \mathbf{x}$ where φ contains relational atoms with the names of relation names contained the relational expression of the R2VML mapping. In the creation of st-tgds, each two rules use mutually disjoint set of variables. In the end, we have a set of st-tgds Σ_{st} where we analyze if the setting is consistent using the Algorithm 4. This algorithm has as inputs

the relational schema $\mathbf{R}$, the shapes schema $\mathbf{S}$, the set of st-tgds Σ_{st} and the set of IRI constructors, and gives as output a Boolean value. $True$ if the setting is consistent, $false$ otherwise. The algorithm is as follows. For any two rules in Σ_{st}, if both rules use the same IRI constructor f for their subjects and have the same predicate, the same subject type T and there is a type $S \in \mathscr{T}$ such that $p :: S^{\mu} \in \delta(T)$ with $\mu = \texttt{1}$ or $\mu = \texttt{?}$, then an instance I is constructed similarly as a contentious-based instance (line 3 to 5). Then, the algorithm chases I with the set of functional dependencies (line 6). Finally, the algorithm checks if the result of the chase has equated two different values (line 7), and if so, the algorithm finishes with a $false$ value. If the algorithm ends without equating two values (line 7), then the setting is consistent.

Algorithm 4: Deciding consistency in ShERML.

Input: a data exchange setting $\mathscr{E} = (\mathbf{R}, \mathbf{S}, \Sigma_{\text{st}}, \mathbf{F})$
Output: $true$ if $\mathscr{E}$ is value consistent,$false$ otherwise.

1 **for** $(\sigma, \sigma') \in \Sigma_{\text{st}} \times \Sigma_{\text{st}}$ **do**
2 **if** $head(\sigma) = T(f(\mathbf{x})) \wedge Triple(f(\mathbf{x}), p, t_1)$ *and* $head(\sigma') = T(f(\mathbf{x})) \wedge Triple(f(\mathbf{x}), p, t_2)$ *and* $\exists S \in \mathscr{T} \,.\, p :: S^{\mu} \in \delta(T)$ *for some* $\mu \in \{\texttt{1}, \texttt{?}\}$ **then**
3 $B_{\sigma,\sigma'} = body(\sigma) \cup body(\sigma')$;
4 $h_{\sigma,\sigma'} : vars(\sigma) \cup vars(\sigma') \rightarrow \mathsf{NullLit}$;
5 $I = h_{\sigma,\sigma'}(B_{\sigma,\sigma'})$;
6 $J = chase(I, \Sigma_{\text{fd}})$ and the corresponding $h : I \rightarrow J$;
7 **if** $(h_{\sigma,\sigma'} \circ h)(t_1) \neq (h_{\sigma,\sigma'} \circ h)(t_2)$ **then**
8 **return** $false$;
9 **end**
10 **end**
11 **end**
12 **return** $true$;

5.5.6 Additional features

Now, we describe three features of ShERML: exporting and importing of VML script, R2VML visualization and color configuration. ShERML allows to export and import VML scripts so a user can continue defining more mappings or reuse the mappings already defined in the VML script with other source instance of the same relational schema and materializes it. R2VML visualization shows the R2VML mappings for those users that are familiar with text-based languages. Users can find R2VML map-

pings in a VML script, but these mappings are not readable because the script combines R2VML mappings with VML mappings and position annotations. The feature of color configuration allows users to assign colors to the three different kinds of arrows.

5.6 Evaluation

In this section, we have assessed our tool in its usability with six users: three of them are undergraduate students of computer science, one user is a doctoral student, one user is a software engineer that works with system databases and the last user is a researcher on the semantic web. We identify these users with the following letters (a) to (c) for the undergraduate students, (d) for the doctoral student, (e) for software engineer and (f) for the researcher.

5.6.1 Methodology

The methodology for testing ShERML consists of three phases: preparation, evaluation and observation. In the preparation phase, we provide every user a manual of how it is used ShERML describing its main functionalities. Also, the manual contains a brief description of what is shapes schema. Then, we provide a testing document containing four cases where we describe every case as follows. A case is presented with a description of the relational database, a screenshot of graph representations of the relational and shapes schema, the set of mappings to perform expressed in natural language and the desired graph that should be obtained when materializing the mappings. Next, we provide two scripts: a SQL script containing the relational schema and the instance, and JSON script containing a shapes schema. Finally, we guide the user in the installation of the application. The tool, all testing scripts, manual and testing document can be found in this repository `htpps://github.com/josemachino/ShERML`.

In the evaluation phase, the user can ask questions while doing the mappings if they do not understand certain functionalities of the tool. If users do not succeed to do the mappings they can leave the case and pass to another one. Before passing to other cases, the user sends us the VML mappings. We do not restrict the time for each case.

Finally, in the observation phase, users provide us their impressions of ShERML and

we observe if their VML mappings are correct and analyze the reason for the incorrect mappings. For this purpose, we distinguish two types of mappings. A mapping is *simple* if the path is composed of less equal than two relational names, otherwise the mapping is *complex*. Case one and two are composed of simple mappings, while cases three and four are composed of complex mappings.

5.6.2 Results

The results of the correct mappings done by a user in the four different cases are shown in Table 5.2. In cases one and two, we observe that users (a), (b) and (e) did not succeed to do the correct mappings because they created property arrows instead of reference arrows. User (c) did the reference arrows but the IRI constructors were incorrect. Users (d) and (f) did correctly mappings.

In cases three and four, we observe that users (a) and (c) did not succeed the complex mappings because the notion of paths is not well understood. One reason can be that the manual only presented a case of simple mappings and did not emphasize the notion of paths. The manual did not describe the notion paths because we were interested in evaluating the interactions of the editor. The result was not as expected because not all undergraduate students, who have less experience in databases, succeeded to make complex mappings. For the rest of users, the notion of paths was understood because they have more experience in databases.

Type of Mapping	**Case**	**Expected number of mappings**	**Correct mappings per user**					
			a	b	c	d	e	f
Simple	1	3	2	2	1	3	3	3
	2	5	2	3	2	5	2	5
Complex	3	3	1	3	1	3	3	3
	4	3	0	3	0	3	3	3

Table 5.2: Correct mappings per case and user.

Now, we analyze the comments of users about ShERML:

- Users (a), (b) and (c): it is not clear the use of the IRI constructor and its purpose. One reason because these users did not have a clear understanding of IRI

constructors is the lack of knowledge about RDF where the domain is over IRIs or literals.

- All users: Many arrows crossing each other in the editor for case two. ShERML cannot arrange the arrows in the editor such that they are not crossed. But in the mapping panel, ShERML presents the corresponding VML mappings without crossing lines. However, a solution for the crossing of arrows in the editor is not possible. This is a drawback of ShERML.

- User (f): Most issues of users with Ontop[2], which is an application that relies on R2RML mappings, comes from bad mapping design. ShERML addresses this problem through the creation of VML mappings and the conversion to R2RML script.

Finally, we draw the following conclusion from the evaluation:

- ShERML does not have the right interactions for showing the user the importance of the IRI constructor. One way to approach the lack of understanding of the IRI constructor is to have a visual object for the IRI constructor and interaction operation over it. As it is now, the IRI constructor is over the IRI arrow or reference arrow as text. According to users, the text over arrows is lost in the editor. One way to solve this problem is the use of a visual object that captures the attendance of a user to check if the IRI constructor corresponds to the shape name or triple constraint that the arrow connects.

- ShERML has to enhance the readability of the graph representation of schemas by a better distribution of nodes and avoiding the crossing of edges in the panel.

- ShERML helps the creation of complex mappings with the path selector, but it should be more intuitive for non-expert users such (a) and (b).

- ShERML facilitates the mapping design.

- ShERML is compatible with third-party applications such as Ontop.

[2] `https://ontop-vkg.org/`

5.7 Discussion and conclusion

We have achieved in this chapter the definition of a visual mapping language. We have developed ShERML that hides for non-expert users the complexity of learning a data exchange framework and creates VML mappings. ShERML is a tool that offers its users an easy mapping process and a distinct characteristic of this tool is the consistency verification. The feature of ShERML of import and export of VML script allows users to reuse the data exchange setting with other instances.

However, ShERML has some limitations inherent to the framework that we have chosen. For instance, we limit the kind of queries that can be expressed with R2VML as discussed in Section 5.3. Also, the aim to produce a solution to a given instance of a relational schema w.r.t. the shapes schema carries the limitation of representing null values that are meaningless for users as discussed in Section 5.5.3.

As future work, we propose the enhancement of some functionalities and considering additional features for a better data exchange experience. We will improve the user feedback in the consistency checking such that a user can know the arrow that is causing the inconsistency of the setting. In terms of additional features, we will add the functionality of searching VML mappings in the visualization panel and a graph visualization of the RDF graph.

5.8 Related work

Different tools [Fagin *et al.* 2009, Raffio *et al.* 2008, Pichler & Savenkov 2009, Marnette *et al.* 2011, Sengupta *et al.* 2013, Sicilia *et al.* 2017] have been developed to assist users in the definition of mappings in relational, XML and relational to RDF data exchange. We describe first those tools used for relational and XML data exchange. Then, we describe the tools used in relational to RDF data exchange.

Clio system [Fagin *et al.* 2009] presents a graphical interface to assist a user in the definition of mappings in the relational data exchange context. It uses a tree representation to show the source and target schemas with constraints. This tree representation does not capture the inclusion dependencies information. The definition of mappings is by drawing arrows between attributes. ShERML uses graph representation for relational

schema where the inclusion dependencies information is also captured. Our tool is used in a relational to RDF data exchange context with shape constraints. This implies the use of IRI constructors and verification of constraints when doing the arrow. Thus, in ShERML is distinguished two arrows more. Also in Clio, SQL is used to present the mapping that is less declarative than R2VML.

Clip [Raffio *et al.* 2008] is a tool used in XML data exchange context. It uses tree representations to show the source and target XML schemas. The definition of mappings is done by drawing of arrows and expressed using XQuery. ShERML is used in another context, but the use of arrows to define mappings is the same approach implemented in our tool.

Pitchler and Savenkov [Pichler & Savenkov 2009] present a data exchange modeling tool called DEMo. In this tool, there is no target schema loaded in the input. They define the target schema while doing the mappings that are SQL queries. This tool does not use visual representations for schemas. The set of mappings defined in this tool is stored in a script so users can continue with the mapping definition later on. We take this idea of storing the set of mappings in a file.

++Spicy [Marnette *et al.* 2011] is a tool for relational and XML data exchange context. The tool uses tree representations for the source and target schema and the mappings are defined with arrows. The tool allows to define filters on the attributes. We take this idea to be used in ShERML by allowing filters on attributes. The tool considers target as ShERML does.

Now, we describe tools used for relational to RDF data exchange. One of them uses D2RQ and the rest use R2RML as the language to define mappings. None of these tools uses target constraints. RDOTE [Vavliakis *et al.* 2013] is a tool that develops a text-based language called D2RQ to define mappings from a relational schema to an ontology. An ontology defines a set of classes that are organized in a hierarchical taxonomy, a set of properties of those classes and relationships between those classes. The input of this tool is a database and an ontology. RDOTE defines D2RQ mappings manually in a editor form.

Sengupta et al. [Sengupta *et al.* 2013] present a R2RML mapping editor. This editor allows the materialization of an RDF graph. Defining a R2RML mapping is done in an editor form. Also, this editor provides a search utility where given a set of VML

mappings defined by the editor, users can search a mapping by the path specify in the SQL query and the term map used in the subject, predicate and object. This utility can be added in ShERML to easily find a VML mapping from the list of VML mappings created by the editor. As it is now, users have to scroll down in the visualizing panel to find a VML mapping. Their tool does not use visual representations.

Heyvaert et al. [Heyvaert *et al.* 2016] present a tool called RML editor. This tool has a visual graph-based user interface, which allows users to create mapping rules graphically. The tool consists of three panels: input panel, modeling panel and result panel. In the input panel, a user loads a relational database that is represented with tabular data. There is no visual representation of a relational schema. Then the user defines IRI constructors in the modeling panel. The visual object for the IRI constructor is a circle that contains its definition. Its definition consists of an IRI with attribute names coming from the relational schema. Then the user drags attributes names from a table and a visual representation is created in the modeling panel for the attribute name. Next, the user draws arrows from the visual representation of an IRI constructor to the visual representation of an attribute name. This arrow is interpreted as a R2RML mapping. The graph defined in the modeling panel defines a set of R2RML mappings that is used in the result panel to show the materialized graph. Similar to this tool, we define IRI constructors but not with a visual object only as text on top of an arrow.

Map-On [Sicilia *et al.* 2017] is a tool that defines a visual mapping language for relational to RDF data exchange with an ontology as a target schema. Map-On uses arrows between elements of the relational schema to elements of the ontology. The visual representation for each element is a box and they use different colors to distinguish table, attribute, and type names of the ontology. The set of arrows in the panel creates R2RML mappings.

Juma [Junior *et al.* 2017] is a tool that creates R2RML mappings using the block metaphor. Each R2RML rule is a block and each block is composed of sub-blocks that construct the rule such a sub-block for the query where the tool gets the data, sub-block for the subject and other for predicate object. This tool only allows the connection of blocks that would create a valid mapping. Juma defines a visual mapping language on top of R2RML using blocks as the visual representation for every element of R2RML. This tool requires that users must known R2RML syntax while in ShERML we hide

the step of learning R2VML. A similar feature to our tool is that we also create valid mappings.

SQuaRE [Bak *et al.* 2017] is a tool that allows to create R2RML mappings. The target schema is an ontology. The user loads a database and an ontology. Because an ontology is a hierarchy of concepts, the tool uses a tree representation for the ontology. There is no visual representation of the relational schema. Instead, the user has to choose a table from the relational and a visual representation only of the table is shown. SQuaRE has a mapping panel where the user drags types of the ontology and draw arrows from the attributes to the types in the mapping panel. A feature of this tool is that they allow conditions on the mappings. From these arrows, a R2RML mapping is generated. This tool has defined a visual mapping language over the relational signature and set of types of the ontology. In ShERML, we also allow to set conditions on the blue arrows and use the arrows to generate mappings.

YARRRML [Heyvaert *et al.* 2018] is a R2RML editor that uses YAML [Ben-Kiki *et al.* 2009] to write R2RML mappings. YAML is a data serialization language that simplifies the elements of R2RML syntax to be more human friendly. Visual representations are not used in YARRRML.

Tools	**VML**	**Text-based language**	**VR for schemas**	**Mapping overview**	**Target constraints**	**Solution**
RDOTE	No	D2RQ	No	No	No	Yes
Sengupta et al.	No	R2RML	No	No	No	Yes
RML editor	Yes	R2RML	No	Yes	No	Yes
SQuaRE	No	R2RML	Yes	Yes	No	Yes
Map-On	Yes	R2RML	Yes	Yes	No	Yes
YARRRML	No	R2RML	No	No	No	Yes
ShERML	Yes	R2VML	Yes	Yes	Yes	Yes

Table 5.3: Summary of tool features.

Finally, Table 5.3 summarizes the main features of the different mapping tools in the relational to RDF data exchange context and of ShERML presented in this chapter. The acronym VR is for visual representation and VML for visual mapping language. The tools are compared with the visual mapping language defined, the text-based mapping language used, whether they use visual representations in their tool, whether they

provide a panel to see all the mappings created by the tool, whether they are concerned with target constraints and if they materialized having a solution. Since most these tools work with an ontology as target schema, they always a produce solution because of the absence of constraints.

Chapter 6

Shapes schema elicitation

6.1 Motivation

The presence of R2RML standard indicates that there is a considerable amount of web applications that exchange information using the RDF format. In this context, the exchanged information comes from relational databases. R2RML mappings might specify the type of the nodes that they produce, but in this chapter, we focus on mappings where no node is typed. Also, we focus on R2RML mappings as modeled by constructive st-tgds. We consider the problem of schema elicitation which means constructing a shapes schema that describes the structure of the output graph.

Schema elicitation is not an easy task. We identify the main primary challenge: the differences between the source and target languages i.e., on one side we have the language for expressing constraints of the source relational databases and on the other side we have the language for expressing target shapes schemas. Also, the set of mappings can have additional information that must be considered in the elicitation of the target schema. As consequence of the difference of languages, there might not exist a target schema that capture all the graphs *produced* by the mapping.

To address this challenge, we characterize what a good target schema is. We identify soundness and completeness as properties of a desirable target schema. A sound schema recognizes every possible output of the mapping. A complete schema recognizes only graphs that can be a possible output of the mapping. Finally, we present an elicitation algorithm that is sound but also that is complete for a large practical set of

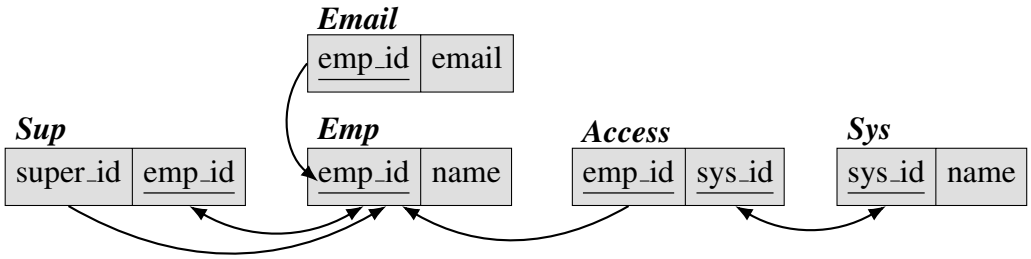


Figure 6.1: A relational schema.

source relational schemas and mappings.

Example 6.1.1. Consider the source schema $\mathbf{R}$ shown in Figure 6.1 where arrows represent the inclusion dependencies.
The following constructive st-tgds are given:

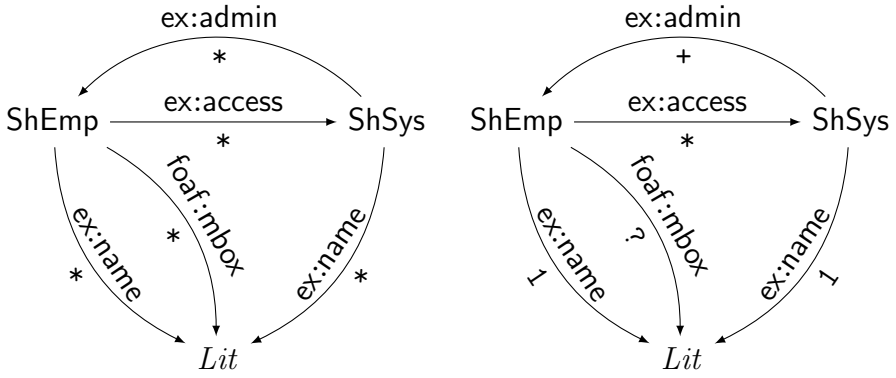
$$\textit{Emp}(x_1, x_2) \Rightarrow \textit{Triple}(f_{emp2iri}(x_1), \mathsf{ex{:}name}, x_2), \tag{6.1}$$

$$\textit{Email}(x_1, x_3) \Rightarrow \textit{Triple}(f_{emp2iri}(x_1), \mathsf{foaf{:}mbox}, x_3), \tag{6.2}$$

$$\textit{Sys}(x_1, x_2) \Rightarrow \textit{Triple}(f_{sys2iri}(x_1), \mathsf{ex{:}name}, x_2), \tag{6.3}$$

$$\textit{Access}(x_1, x_2) \Rightarrow \textit{Triple}(f_{emp2iri}(x_1), \mathsf{ex{:}access}, f_{sys2iri}(x_2)), \tag{6.4}$$

$$\textit{Access}(x_1, x_2) \wedge \textit{Sup}(x_3, x_1) \Rightarrow \textit{Triple}(f_{sys2iri}(x_2), \mathsf{ex{:}admin}, f_{emp2iri}(x_3)). \tag{6.5}$$



(a) Sound and not complete (b) Sound and complete

Figure 6.2: Shapes schemas elicited.

We examine the set of st-tgds and observe the IRI constructors. We propose for every IRI constructor a dedicated shape name: ShEmp for $f_{emp2iri}$ that represents employees and ShSys for $f_{sys2iri}$ that represents systems. We can propose two schemas as seen in Figure 6.2, where one of them is sound and the other is sound and complete. The schema

in Figure 6.2a is obtained by simply identifying the outgoing edges of ShEmp and ShSys and the multiplicity for every edge be $*$. For instance, there is an outgoing edge with label ex:name from ShEmp to Lit because in the mapping 6.1 no IRI constructor is used in the object position. Also, we observe the outgoing edge ex:admin that connects ShSys with ShEmp. This corresponds to the mapping 6.5 because the IRI constructor $f_{sys2iri}$ appears in the subject position and the IRI constructor $f_{emp2iri}$ in the object position. This target schema is sound but it is overly general and not complete because this schema allows to have a shape employee with two emails or no name, and yet this kind of information is not allowed by the relational schema.

By carefully analyzing the mappings and the set of dependencies of the relational schema we can refine the previous schema to obtain a sound and complete schema as shown in Figure 6.2b. This refinement is done by precisely determining the multiplicity constraints, which are obtained from the analysis of the set of source dependencies and the mapping. Thus, we will make a better fit for the shapes schema. For the outgoing edge ex:name from ShEmp and ShSys, the multiplicity constraint is 1 because there is a key dependency that restricts to have exactly one name for an employee and system. For the outgoing edge foaf:mbox from ShEmp, we observe the relation *Email* has an inclusion dependency that references to an employee and the attribute emp_id is a primary key for *Email*. This means that an employee will have at most one email and sometimes might have none. Thus, the multiplicity is ? for this outgoing edge. For the outgoing edge ex:access from ShEmp, we observe that the primary key of *Access* is composed of the attributes emp_id and sys_id with inclusion dependencies to ShEmp and *Sys*, which means that an employee might have access to multiple systems and might have access to none. Thus, the multiplicity is $*$. Now the rule that indicates that an employee is an admin of a system is (6.5). Here, we observe the inclusion dependencies related to *Access* and *Sup*. With respect to *Access* the inclusion dependencies shown in Figure 6.1 indicates that a system must be accessed by at least one employee. With respect to *Sup*, the inclusion dependencies shown in Figure 6.1 indicates that an employee must have a supervisor. From these two observations, we conclude that for the outgoing edge ex:admin, the multiplicity is +. □

Also, we show that there are cases where there might not exist a target schema that

is complete or if a complete a schema exists then this schema is of exponential size. In the next example, we in fact show that a complete target schema might not exist.

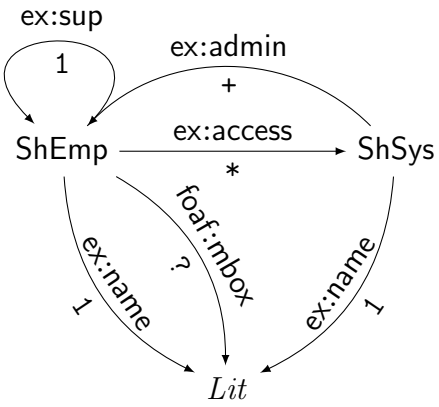


Figure 6.3: Sound and not complete.

Example 6.1.2. Now, we consider the additional st-tgd that connects employees with their supervisors.

$$\mathit{Sup}(x_1, x_2) \Rightarrow \mathit{Triple}(f_{emp2iri}(x_1), \mathsf{ex{:}sup}, f_{emp2iri}(x_2)).$$

Following the same procedure of elicitation, we construct the schema shown in Figure 6.3. This schema is sound, however, it is not complete. We observe the graphs

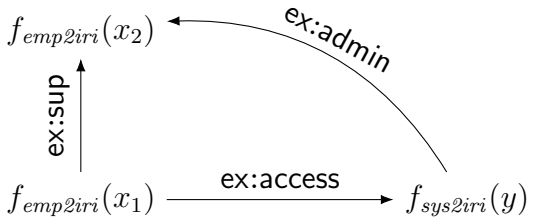


Figure 6.4: A triple pattern present in any output of Σ.

obtained from the application of this new set of st-tgds have a triangle pattern shown in Figure 6.4. Consequently, a complete target schema does not exist because the language of shapes schema cannot express such a triangle pattern. Here, we observe that the differences between the source and target languages for expressing constraints together with the information coming from the mappings renders elicitation of a complete shapes schema unfeasible. □

6.2 Problem statement

In this section, we define formally the elicitation problem that takes a relational to RDF data exchange setting without schema and outputs a target suitable shapes schema and we propose two desirable properties of target schema.

6.2.1 Schema-less data exchange setting

We define a schema-less data exchange setting that we use as input for the elicitation problem.

Definition 6.2.1. A *schema-less relational to RDF data exchange setting* is a tuple

$$\mathscr{E}_d = (\mathbf{R}, \Sigma_{\text{st}}, \mathbf{F})$$

such that for every relation R of $\mathscr{R}$, R has a key dependency and the tuple $(\mathbf{R}, \mathbf{S}_\emptyset, \Sigma_{\text{st}}, \mathbf{F})$ is a constructive relational to RDF data exchange setting with $\mathbf{S}_\emptyset = (\emptyset, \emptyset)$ being an empty shapes schema. □

In the sequel, we treat $\mathscr{E}_d$ as the corresponding constructive relational to RDF data exchange setting. In particular by $sol_{\mathscr{E}_d}(I)$ we denote the set of solutions to $I \in Inst(\mathbf{R})$ w.r.t. $\mathscr{E}_d$ i.e., $sol_{\mathscr{E}_d}(I) = \{G \mid I \cup G \models \Sigma_{\text{st}}\}$. Also, by $T(\mathscr{E}_d)$ we denote all graphs *produced* by $\mathscr{E}_d$ over any instance of the relational schema $\mathbf{R}$ i.e.,

$$T(\mathscr{E}_d) = \{sol_{\mathscr{E}_d}(I) \mid I \in Inst(\mathbf{R})\}.$$

Non-deterministic shapes schemas under close interpretation

In previous chapters of this manuscript, we have considered deterministic shapes schemas interpreted under open interpretation. However, in this chapter, we drop the restriction of determinism. More precisely, a shapes schema now can have a type whose shape definition has multiple triple constraints with the same property. For instance, the following shapes schema is non-deterministic because for type TUniv there are two triple

constraints with ex:has.

$$\mathsf{ShUniv} \rightarrow \mathsf{ex{:}has} :: \mathsf{TEmp}^{1}; \mathsf{ex{:}has} :: \mathsf{ShSys}^{+}.$$
$$\mathsf{ShEmp} \rightarrow \mathsf{ex{:}name} :: Lit^{1}; \mathsf{ex{:}sup} :: \mathsf{ShAdmin}^{1}.$$
$$\mathsf{ShAdmin} \rightarrow \mathsf{ex{:}name} :: Lit^{1}; \mathsf{ex{:}sup} :: \mathsf{ShAdmin}^{?}.$$
$$\mathsf{ShSys} \rightarrow \mathsf{ex{:}id} :: Lit^{1}.$$

Also, we consider the multiplicity 0 with its corresponding interval $[0, 0]$ in the multiplicities of a non-deterministic shapes schema.

Now, we define how a graph satisfies a non-deterministic shapes schema under close interpretation. We recall the definition of embedding [Staworko & Wieczorek 2019]. Recall that we view a shape graph as (p, μ)-labeled graph. First, we identify the set of all outgoing edges of a node $n \in nodes(G)$ with

$$out_G(n) = \{(n, p, m) \in G\}.$$

Also, we define a function $occur$ over a graph G that assigns an interval to an edge in G. If the graph is an RDF graph the interval assigned is 1, otherwise, the interval is the multiplicity contained in the label of the edge. Next, we define the point-wise addition operator $\oplus$ between two intervals as follows: $[n_1; m_1] \oplus [n_2; m_2] = [n_1 + n_2; m_1 + m_2]$. Now, we define an embedding as follows:

Definition 6.2.2 (Embedding)**.** Given a RDF graph G and a shape graph H, a binary relation $R \subseteq nodes(G) \times nodes(H)$ is a simulation of G and H iff for any $(n, n') \in R$ we have that

- n is a literal node iff n' is a literal type,
- n is a non-literal node iff n' is a non-literal type, and
- there exists a witness of simulation of n by n' w.r.t. R i.e. a function $\lambda_{n,n'} : out_G(n) \rightarrow out_H(n')$ such that for every $(n, p, m) \in out_G(n)$
  - there is $m' \in nodes(H)$ and $\mu \in \{0, 1, ?, +, *\}$ such that $(n', (p, \mu), m') \in out_H(n')$, $\lambda_{n,n'}((n, p, m)) = (n', (p, \mu), m')$ and $(m, m') \in R$,

– for every $(n', (q, \mu), m') \in H$ it holds that

$$\bigoplus\{occur_G((n, q, o)) \mid (n, q, o) \in G, \lambda((n, q, o)) = (n', (q, \mu), m')\} \subseteq occur_H((n', (q, \mu), m')).$$

An *embedding* of G in H is a simulation R of G in H such that $adom(R) = nodes(G)$, and we write $G \preccurlyeq H$ if G can be embedded in H. □

A graph G satisfies a shapes schema $\mathbf{S}$, if there is an embedding from G into the shape graph of $\mathbf{S}$, i.e., $G \preccurlyeq G_{\mathbf{S}}$. We illustrate graphically an embedding in the following example.

Example 6.2.3. Consider the shapes schema $\mathbf{S}$ presented previously and the graph G presented in the left hand side of Figure 6.5, with the interval 1 added to each edge for the purpose of illustrating the embedding. In the right hand side, we have the shape graph of the $\mathbf{S}$. This graph satisfies the non-deterministic shapes schema because there is embedding as shown in Figure 6.5. □

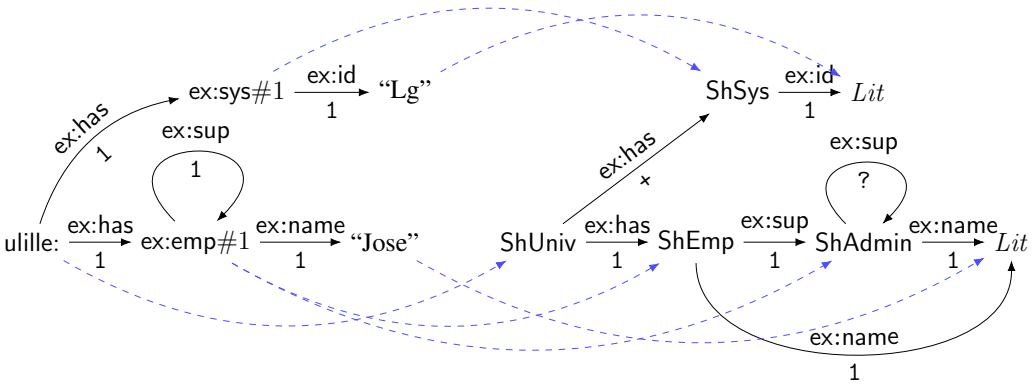


Figure 6.5: An example of embedding.

Finally, we define the language of a non-deterministic shapes schema under close interpretation as the set of graphs that can be embedded in the shape graph of the shapes schema i.e.,

$$L_{\mathsf{C}}(\mathbf{S}) = \{G \mid G_{\mathbf{S}} \text{ is the shape graph of } \mathbf{S}, G \preccurlyeq G_{\mathbf{S}}\}.$$

6.2.2 Elicitation Problem

The *schema elicitation problem* aims at constructing a target schema for a given relational to RDF schema-less data exchange setting. We identify two desirable properties of the target schema. The first property is *soundness* that requires the target schema to recognize every output of the mapping. This property is not enough because trivial solutions do not inspect schemas and obtain universal schemas, which are also sound. The second property is *completeness*, which intuitively means that the target schema recognizes only graphs that can be produced by the mapping. Formalizing completeness is challenging because graphs produced by mappings use node identifiers that are generated by IRI constructors and those may have limited range. We circumvent this problem by employing the standard notion of isomorphism.

Example 6.2.4 (cont. Example 6.1.1.)**.** In this example, we assume that for an employee the IRI constructor is

$$f_{emp2iri}(id) = \text{“}\texttt{univ:emp\#}\text{”} + \texttt{str}(id)$$

and for a system, the IRI constructor is

$$f_{sys2iri}(id) = \text{“}\texttt{univ:sys\#}\text{”} + \texttt{str}(id).$$

Take a graph G_0 recognized by the language of $\mathbf{S}$ shown in Figure 6.2b. The identifier that we use in the graph G_0 cannot be obtained by the IRI constructors described above. From G_0, we construct an instance of the relational schema as presented in Figure 6.6. If we apply the mappings with Σ_{st}, then we get an isomorphic graph as shown on the right side in Figure 6.6. Here, we observe that both graphs are recognized by the language of the schema shown in Figure 6.2b. □

Formally, two graphs G and G' are *isomorphic*, denoted by $G \cong G'$, if there is a bijective homomorphism from G to G'. Also, for two set of graphs $\mathscr{G}$ and $\mathscr{H}$, we write $\mathscr{G} \subseteq_{\text{iso}} \mathscr{H}$ iff for any graph $G \in \mathscr{G}$, there is a graph $G' \in \mathscr{H}$ such that G and G' are isomorphic. Now, we can formally state the two desirable properties as follows.

Definition 6.2.5. Given a schema-less data exchange setting $\mathscr{E}_d$, a target schema is

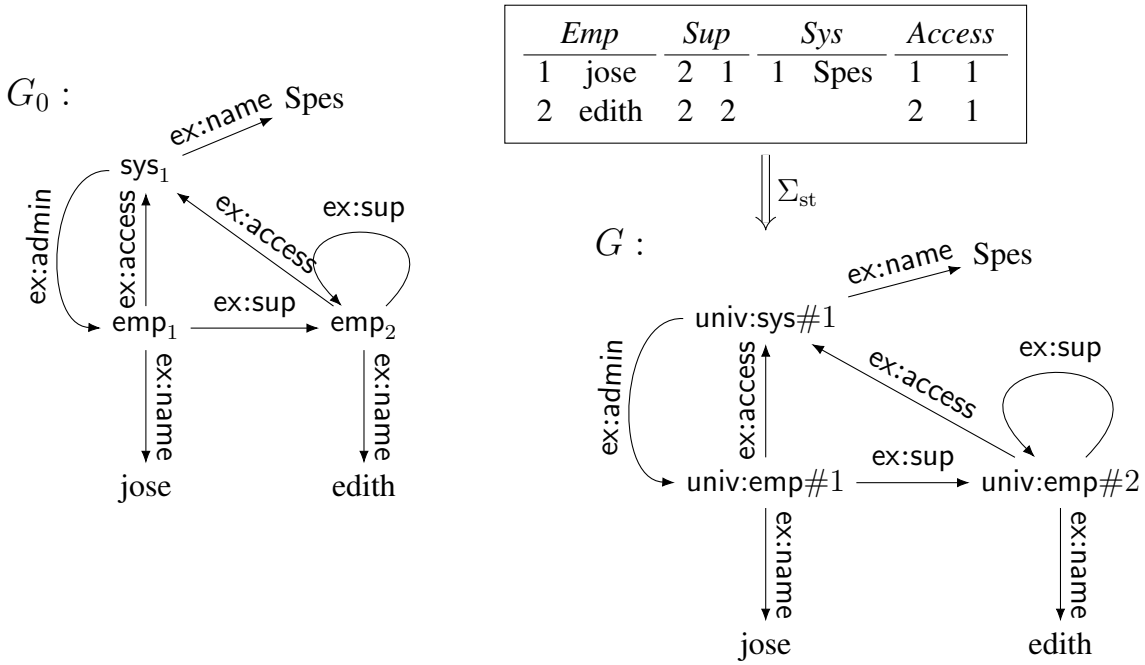


Figure 6.6: Example of graphs that are isomorphic.

1. *sound* w.r.t. $\mathscr{E}_d$ iff all graphs produced by the setting are recognized by the language of the schema under close interpretation i.e., $T(\mathscr{E}_d) \subseteq L_{\mathsf{C}}(\mathbf{S})$, and

2. *complete* w.r.t. $\mathscr{E}_d$ iff for any graph that a schema recognizes a mapping can produce a isomorphic one, in symbols $L_{\mathsf{C}}(\mathbf{S}) \subseteq_{\text{iso}} T(\mathscr{E}_d)$. □

In the sequel, we fix a schema-less data exchange setting $\mathscr{E}_d = (\mathbf{R}, \Sigma_{\text{st}}, \mathbf{F})$.

6.3 M3 Elicitation algorithm

In this section, we propose an elicitation algorithm called *min-max models* (M3), which constructs the minimum and maximum models that describe the minimum and maximum multiplicities in the output schema. For every IRI constructor f, the algorithm constructs a type T_f of nodes constructed by f obtaining that the set of types $\mathscr{T}$ for the target schema is defined as follows:

$$\mathscr{T} = \{T_f \mid f \in \mathscr{F}\}.$$

We illustrate the M3 algorithm with the following example.

Example 6.3.1. Consider the schema-less setting where the relational schema has two relations $R(\underline{x}, y), P(\underline{x, y})$ and the inclusion dependency $P[x] \subseteq R[x]$. The set of mappings Σ_{st} is

$$R(x, y) \Rightarrow \mathit{Triple}(f(x), r, y), \tag{6.6}$$

$$P(x, y) \Rightarrow \mathit{Triple}(f(x), p, y). \tag{6.7}$$

First, for each mapping, we construct the minimal model as follows. We take mapping (6.6) and construct from it a canonical instance $I_1 = \{R(x, y)\}$. We observe it is consistent with $\mathbf{R}$. Then, we apply the mappings and obtain the graph

$$G^1_{\text{min}} = \{\mathit{Triple}(f(x), r, y)\}. \tag{6.8}$$

From G^1_{min}, we construct a *shape expression* for T_f, which is a set of triple constraints. So we observe that G^1_{min} has a triple with $f(x)$ and r and, therefore, there must be a triple constraint with label r and multiplicity `1`. Because in the set of mappings there is a triple atom with p and this graph has no outgoing edges with p from $f(x)$, there must be a triple constraint with p and multiplicity `0`. We obtain the following shape expression that fits this particular graph:

$$r :: \mathit{Lit}^{\mathtt{1}}; p :: \mathit{Lit}^{\mathtt{0}}$$

We do an analysis later on over the corresponding shape expression. Now, we process the mapping (6.7) and construct the canonical instance $I_2 = \{P(x, y)\}$. We notice that I_2 is not consistent with $\mathbf{R}$ because of the inclusion dependency $P[x] \subseteq R[x]$. Therefore, we chase it with $\Sigma_{\mathbf{R}}$ obtaining $I'_2 = \{P(x, y), R(x, \perp)\}$. Next, we apply the mappings obtaining the graph

$$G^2_{\text{min}} = \{\mathit{Triple}(f(x), r, \perp), \mathit{Triple}(f(x), p, y)\} \tag{6.9}$$

Upon inspection, G^2_{min} yields shape expression for T_f

$$r :: \mathit{Lit}^{\mathtt{1}}; p :: \mathit{Lit}^{\mathtt{1}}$$

Now, we proceed to construct the maximal model as follows. We take the graph produced by all mappings with f in the subject

$$G^{f}_{\max} = \{\, \mathit{Triple}(f(x), r, \bot_1),\, \mathit{Triple}(f(x), p, \bot_2)\}.$$

Next, we *saturate* it by duplicating each outgoing edge label with fresh null values for the target nodes. The result is the following graph

$$G'^{f}_{\max} = \{\, \mathit{Triple}(f(x), r, \bot_1),\, \mathit{Triple}(f(x), r, \bot_3),\, \mathit{Triple}(f(x), p, \bot_2),\, \mathit{Triple}(f(x), p, \bot_4)\}.$$

Then, we apply backwards the mappings Σ_{st} to $G'^{f}_{\max}$ obtaining the following instance

$$I_{\max} = \{R(x, \bot_1), R(x, \bot_3), P(x, \bot_2), P(x, \bot_4)\}.$$

Here, we observe that $I_{\max}$ satisfies the inclusion dependencies, but the functional dependency $R : x \rightarrow y$ is not satisfied. Thus, we perform the chase, which equates the null values $\bot_1$ and $\bot_3$, yielding $I'_{\max} = \{R(x, \bot_1), P(x, \bot_2), P(x, \bot_4)\}$. Then, we apply the mappings to I'_0 yielding a graph

$$G''^{f}_{\max} = \{\, \mathit{Triple}(f(x), r, \bot_1),\, \mathit{Triple}(f(x), p, \bot_2),\, \mathit{Triple}(f(x), p, \bot_4)\}. \tag{6.10}$$

We observe the multiplicities related to $f(x)$ in $G''^{f}_{\max}$, we restrict the maximum multiplicity to be 1 for label r and infinite for label p obtaining the following shape expression for T_f

$$r :: \mathit{Lit}^{1}; p :: \mathit{Lit}^{+}$$

Finally, we obtain the shape definition for T_f by fitting the shape expressions obtained from the minimal models (6.8), (6.9) and the maximal model (6.10). The result of this fitting is

$$T_f \rightarrow r :: \mathit{Lit}^{1}; p :: \mathit{Lit}^{*}$$ □

6.3.1 Preliminary notions

We recall the notion of canonical instance and introduce the terminology of minimal and maximal models. We also modify the chase procedure for purpose of construction of the minimal and maximal models.

Canonical instance

By *canonical source instance* of a constructive st-tgd of the form $\varphi(\mathbf{x}, \mathbf{y}) \Rightarrow \psi(\mathbf{y})$, we mean an instance whose facts are the relational atoms of φ with the set of variables used in the subject term treated as constants and the rest of variables are assigned to fresh null values. For instance, for the st-tgd

$$\varphi(x, y, z) = \textit{User}(x, y) \wedge \textit{Email}(x, z) \Rightarrow \textit{Triple}(f(x), p, y)$$

the corresponding canonical instance is $I_\varphi = \{\textit{User}(x, \perp_1), \textit{Email}(x, \perp_2)\}$.

Localized chase for source dependencies

The problem of using the standard chase procedure for our purposes are potential infinite chase sequences due to inclusion dependencies. In the context of schema elicitation, we are interested in the triples produced for a specific subject and, moreover, we have a bound on how many similar triples are required to define the maximum multiplicity of a triple constraint. Consequently, we can stop pursuing a branch of chase with inclusion dependencies when we determine it will not add further information relevant to the task at hand. We, therefore, propose a *localized* chase procedure, which we first illustrate in the following example.

Example 6.3.2. Consider the relational schema with three relations $R(\underline{x}, y)$, $P(\underline{x, y}, z)$, $Q(\underline{x, y})$ and the set of inclusion dependencies and the corresponding tgds.

$$R[x\,y] \subseteq P[y\,z] \qquad R(x,y) \Rightarrow \exists z'.\, P(z',x,y) \tag{6.11}$$

$$P[x\,y] \subseteq Q[x\,y] \qquad P(x,y,z) \Rightarrow Q(x,y) \tag{6.12}$$

$$P[z] \subseteq R[y] \qquad P(x,y,z) \Rightarrow \exists y'.\, R(y',z) \tag{6.13}$$

$$P[y] \subseteq R[y] \qquad P(x,y,z) \Rightarrow \exists y'.\, R(y',y) \tag{6.14}$$

$$Q[y] \subseteq P[y] \qquad Q(x,y) \Rightarrow \exists y', y''.\, P(y',y,y'') \tag{6.15}$$

The set of mappings Σ_{st} is

$$R(x,y) \Rightarrow \mathit{Triple}(f(x), r, y), \tag{6.16}$$

$$Q(x,y,z) \Rightarrow \mathit{Triple}(f(x), q, y), \tag{6.17}$$

$$P(y,x,z) \Rightarrow \mathit{Triple}(f(x), p, x). \tag{6.18}$$

We intentionally rename the variables so that the same variable is used in the IRI constructor and this variable allows to distinguish *target positions* in the bodies of the mappings that may yield a node constructed with the IRI constructor f. For mapping (6.16), we take the source canonical instance $I = \{R(x, \bot_0)\}$. The standard chase procedure on I with Σ_{ind} is infinite because there is a cycle of triggering inclusion dependencies (6.11), (6.14), (6.11), obtaining an instance at step three $I^* = \{R(x, \bot_0), P(\bot_1, x, \bot_0), R(\bot_2, x), P(\bot_3, \bot_2, x), \ldots\}$. Thus, we use the localized chase that focuses on the IRI constructor f and the *set of trace variables* $A = \{x\}$.

First, we construct the *value flow dependency graph* G^f_A of Σ_{ind} w.r.t. Σ_{st} in Figure 6.7. Its nodes are positions in relational names such as

$$\mathit{nodes}(G^f_A) = \{(R,1), (R,2), (P,1), (P,2), (P,3), (Q,1), (Q,2)\},$$

and edges identify values that can be transferred from one position to other with the application of an inclusion dependency. Then, we identify the distinguished target positions in relational atoms of the mappings Σ_{st} that may yield a triple with $f(x)$ as its subject. For ease of reference, we assign a unique label to each edge in the graph.

Now, we perform localized chase on $I = \{R(x, \bot_0)\}$. We first trigger the tgd (6.11)

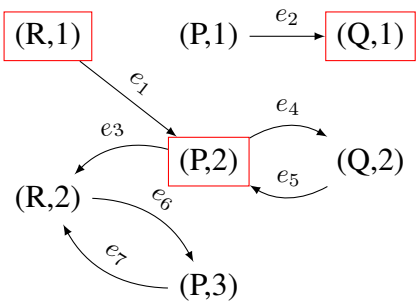


Figure 6.7: Value flow dependency graph.

because its body is on I and its head is not and it is *productive*, i.e.,the edge e_1 takes the value x that appears in the position $(R, 1)$ in the instance I and leads the value to the marked position $(P, 2)$. We apply this tgd obtaining $I_1 = \{R(x, \perp_0), P(\perp_1, x, \perp_0)\}$ and we mark this tgd to prevent triggering it multiple times. Here, for the two inclusion dependencies (6.13) and (6.14), the body is in I_1 but not the head is, yet we shall not trigger them. Consider the tgd (6.14), the corresponding edge is e_3 in Figure 6.7, which indicates that this tgd can send the values from position $(P, 2)$ to $(R, 2)$. We observe that from position $(R, 2)$ there is a cycle that does not pass by a marked a position. Therefore, this tgd is not productive and we do not run it. Now, consider tgd (6.13), it is not productive because no path starts with e_7 that leads to a marked position. We point out that the localized chase is defined so that if the same tgd is in a cycle, then it is not triggered more than once as we see for the next tgd. The third tgd (6.12) has not been triggered before and it is productive because the path e_4, e_5, which is a cycle, leads to a marked position. Thus, we apply it obtaining the instance $I_2 = \{R(x, \perp), P(\perp_1, x, \perp_0), Q(\perp_1, x)\}$ and we mark this tgd. Then, the tgd (6.15) is triggered on I_2 and applied to it obtaining the instance $I_3 = \{R(x, \perp_0), P(\perp_1, x, \perp_0), Q(\perp_1, x), P(\perp_2, x, \perp_3)\}$. Here, we do not trigger the inclusion dependency (6.12) any further because it is marked. Then, there are no more inclusion dependencies to trigger. The result of the localized chase on $I = \{R(x, \perp_0)\}$ with the target $f(x)$ is the *locally-chased canonical instance* I_3 of mapping (6.16). □

The localized chase, denoted by $\ell\text{-}chase$, has as inputs an instance I of $\mathscr{R}$, a set of inclusion dependencies Σ_{ind} of the relational schema, set of mappings Σ_{st}, an IRI constructor f and the set of *trace variables* A used as arguments of f. Because A and the terms used in I ($adom(I)$) contain variables and null values, the constructor cannot

be applied to them. Instead, we shall represent their result symbolically as terms of the form $f(x)$.

We construct the *value flow dependency graph*, denoted by G_A^f, which is a directed graph with distinguished target nodes defined as follows. The nodes are the pairs (R, i) where $R \in \mathscr{R}$ of arity k and $1 \leq i \leq k$, and (R, i) denotes the i-th argument of relation R. We call each pair a position. For every $\rho \in \Sigma_{\text{ind}}$, there is a corresponding set of edges E_ρ defined as follows (we treat inclusion dependencies as tgds):

$$E_\rho = \{((R,i),(P,j)) \in nodes(G_A^f)^2 \mid \mathbf{t}, \mathbf{t}' \subseteq vars(\rho).\, R(\mathbf{t}) \in body(\rho),$$
$$P(\mathbf{t}') \in head(\rho), \exists x \in vars(\rho).\, x \text{ on position } i \text{ of } \mathbf{t} \text{ and on position } j \text{ of } \mathbf{t}'\}.$$

For instance, if we take the inclusion dependency (6.11), we obtain the edges $(R, 1)$ to $(P, 2)$ and $(R, 2)$ to $(P, 3)$ that indicates that values in the sources positions are transferred to the target positions. The set of edges of G_A^f is $E = \bigcup_{\rho \in \Sigma_{\text{ind}}} E_\rho$.

Given a mapping $\sigma \in \Sigma_{\text{st}}$ whose IRI constructor is f, we distinguish a *target position* (R, i) if there is a variable x occurring in the i-argument of R in the body of σ and x is in vocabulary of the subject term of σ i.e., $x \in Vocab(sub(\sigma)) \setminus \mathscr{F}$, where sub is a function that obtains the subject term of a mapping. Formally, the set of marked positions $N_m \subseteq nodes(G_A^f)$ for an IRI constructor f is defined as

$$N_m(f) = \{(R,i) \in nodes(G_A^f) \mid \exists \sigma \in \Sigma_{\text{st}}.\, constructor(\sigma) = f.\, \exists x \in vars(\sigma),$$
$$\mathbf{t} \subseteq vars(\sigma).\, R(\mathbf{t}) \in body(\sigma), x \in Vocab(sub(head(\sigma))),$$
$$x \text{ is in position } i \text{ of } \mathbf{t}\},$$

where $constructor : \Sigma_{\text{st}} \to \mathscr{F}$ is the function that returns the IRI constructor used in the subject position of a st-tgd.

The *localized* chase is used only with inclusion dependencies. For each inclusion dependency ρ, we call a variable x a *transfer variable* if x appears in $body(\rho)$ and in $head(\rho)$. We say that a transfer variable x is *relevant* in the presence of I if there is a homomorphism $h : body(\rho) \to I$ such that $h(x)$ is a trace variable i.e., $h(x) \in A$. At each step, ρ is triggered if ,

- there is a homomorphism h from $body(\rho)$ to the instance,
- there is no extension h' of h such that $h'(head(\rho)) \subseteq I$, and
- there is an edge in E_ρ that begins at a relevant transfer variable and can be extended to a path that leads to a target position in G_A^f.
- ρ has not been triggered more than once i.e., the inclusion dependency is not marked.

The localized chase applies iteratively any triggered inclusion dependency and marks it. The complexity of testing if there exists a trigger for a given instance I and an inclusion dependency is given by the following proposition:

Proposition 6.3.3. Given an instance I of $\mathscr{R}$, testing whether there is an inclusion dependency that is triggered can be done in polynomial time in the size of I.

Proof. Take an inclusion dependency $\rho \in \Sigma_{\text{ind}}$ of the form $R(\mathbf{x}, \mathbf{y}) \Rightarrow \exists \mathbf{z}.\, P(\mathbf{y}, \mathbf{z})$. The first condition can be verified by checking row by row in the instance if it satisfies the body of the dependency. We also need to consider if the head of the dependency is not satisfied in the instance by inspecting row by row. These tasks can be carried out in polynomial time in the size of I. For the third condition, we can use a standard graph reachability algorithm to know if a position leads to a marked position. This task is also done in polynomial time. Thus, testing if an inclusion dependency is triggered is in polynomial time in the size of I. □

The complexity of the localized chase is given by the following proposition

Proposition 6.3.4. Let m be the maximum number of atoms in a st-tgd in Σ_{st}. Let n be the number of inclusion dependencies of Σ_{ind}. Let k be the maximum arity of a relational name in $\mathscr{R}$. Given an instance I bounded by m, the localized chase can be done in polynomial time in $n \cdot k + m$ and its result is bounded by $n \cdot k + m$.

Proof. Take an instance I of $\mathscr{R}$ bounded by m, which is the maximum number of atoms in a st-tgd in Σ_{st}. Let n be the number of inclusion dependencies. The localized chase tests if a dependency is triggered at most n number of times because we do no trigger any inclusion dependency more than once. At each chase step, we add at most one

atom, which varies in the arity of a head of the dependency. This arity is bounded by the maximum arity of a relational name in $\mathscr{R}$. Therefore, the result of the localized chase is bounded by $n \cdot k + m$. Also, at each step by Proposition 6.3.3, we require a polynomial time in the size of I. Then, the localized chase is in polynomial time in $n \cdot k + m$. □

Additional notions

To describe next sections, we define the following auxiliary functions. We assume that nodes of the minimal and maximal models have implicit types and we define the function $node\text{-}type$ that identifies the type of a node as follows:

$$node\text{-}type(n) = \begin{cases} T_g, & \text{if } n = g(x) \text{ for some } x \in \mathscr{V}_\mathscr{Y} \\ Lit, & \text{if } n = x \text{ for some } x \in \mathscr{V}_\mathscr{Y} \end{cases}$$

For instance, given the graph $G = \{Triple(f(x), p, y)\}$, the function applied on the nodes of G obtains $node\text{-}type(f(x)) = f$ and $node\text{-}type(y) = Lit$.

In the sequel, we fix $f \in \mathscr{F}$. For constructing both minimal and maximal model, we denote by Σ_{st}^f the subsets of Σ_{st} that share the same IRI constructor f in their subject terms. Implicitly, we rename the variables so that in all st-tgds of Σ_{st}, the same set of variables appears in the subject term. Every variable that does not appear in the subject term is a *non-shared* variable.

A *locally-chased canonical instance* of a st-tgd $\sigma \in \Sigma_{\text{st}}^f$, denoted by I_σ, is the instance obtained from chasing locally with the source canonical instance of σ and the set of source dependencies $\Sigma_{\mathbf{R}}$ focusing in the IRI constructor f and its arguments. For instance in the Example 6.3.2, I_3 is the locally-chased canonical instance of mapping (6.16) on I.

We define the function multiplicity-set $Mult : 2^{\mathsf{Iri}} \to \{\mathtt{0}, \mathtt{1}, \mathtt{+}\}$ that returns a multiplicity depending on the cardinality of the set $A \in 2^{\mathsf{Iri}}$ defined as

$$mult(A) = \begin{cases} \mathtt{0}, & \text{if } |A| = 0 \\ \mathtt{1}, & \text{if } |A| = 1 \\ \mathtt{+}, & \text{otherwise} \end{cases}$$

A shape expression is a function $E : \mathsf{Iri} \times \mathscr{T} \to \{0, 1, ?, +, *\}$ that assigns to a pair of predicate and type a multiplicity. We define the function $ShExpr(G, n)$ that returns the shape expression E that describes the neighborhood of a node n in graph G and is defined as follows $E(p, t) = Mult(Dst^n_{p:t}(G))$, where $Dst^n_{p:t}(G)$ is the set of target nodes for a given source node n and predicate p, whose type of those nodes is t i.e., $Dst^n_{p:t}(G) = \{m \mid (n, p, m) \in G, \textit{node-type}(m) = t\}$.

We recall that every multiplicity corresponds to an interval. Also, every interval $[n; m]$ has its lower bound $min([n; m]) = n$ and its upper bound $max([n; m]) = m$. We extend the lower bound to a set of intervals $min(Iv) = \{min(k) \mid k \in Iv\}$, and the upper bound to a set of intervals $max(Iv) = \{max(k) \mid k \in Iv\}$. A fitting of a set of basic intervals Iv is defined as follows.

$$\textit{fit}(Iv) = [min(min(Iv)), max(max(Iv))]$$

The result is an interval that is a basic interval.

A *fitting of a set of shape expressions* $\mathbb{E}$ constructs a shape expression E_{fit} from the set of shape expressions as follows. For every property and type $(p, t) \in \mathsf{Iri} \times \mathscr{T}$, E_{fit} is defined as

$$E_{\text{fit}}(p, t) = \textit{fit}(\{E(p, t) \mid E \in \mathbb{E}\}).$$

Minimal models

The *minimal model* of $\sigma \in \Sigma^f_{\text{st}}$ is a graph G^σ_{min} produced by Σ^f_{st} on the locally-chased canonical instance of σ. Recall that M3 algorithm is constructing a single type for every IRI constructor and, consequently, we consider G^σ_{min} a minimal model for this type.

The complexity of constructing all minimal models for a type T_f is given by the following proposition:

Proposition 6.3.5. Let m be the maximum number of atoms in a st-tgd in Σ^f_{st}. Let l be the number of st-tgds in Σ^f_{st}. There is at most l minimal models and the construction of each minimal model is of polynomial size in $n \cdot k + m + l$ where n is the number of inclusion dependencies and k is the maximum arity of relational name in $\mathscr{R}$.

Proof. Let l be the number of st-tgds in Σ^f_{st}. Since the subject term of each st-tgd is

in every minimal model and there is at most l subject terms, then there are at most l minimal models. The construction of the minimal model is the result of chasing the canonical source instance that is bounded by m, which is the maximum number of atoms in a st-tgd in Σ^f_{st}. First, we chase locally, which by Proposition 6.3.4 is polynomial in the size of $n \cdot k + m$ where n is the number of inclusion dependencies and k is is the maximum arity of relational name in $\mathscr{R}$. The result is the locally-chased canonical instance. The chase with functional dependencies does not increase the size of locally-chased canonical instance. On the other hand, at each chase step of a st-tgd, we add one triple atom and since we have l number of st-tgds and by Beeri and Vardi [Beeri & Vardi 1984], then the result is bounded by the size of the input instance then the minimal model is of polynomial size in $n \cdot k + m + l$. □

Maximal model

To obtain the maximal model, we introduce the following terminology. A *saturated instance of a st-tgd* $\sigma \in \Sigma^f_{\text{st}}$, denoted by I^*_σ, is obtained by the following process. We create a copy of σ, denoted by σ', such that both st-tgds share the same variables in the subject term and every non-shared variable is renamed. Then, we compute its locally-chase canonical instance I_σ and $J_{\sigma'}$ for σ and σ' respectively. Finally, we chase the union of these two locally-chase canonical instances with the functional dependencies of $\mathbf{R}$, $I^{**}_\sigma = chase(I_\sigma \cup I_{\sigma'}, \Sigma_{\text{fd}})$. A *saturated instance for* f, denoted by $I^f_{\max}$, is the instance obtained from the union of saturated instances of st-tgds in Σ^f_{st}, $I^f_{\max} = \bigcup\{I^{**}_\sigma \mid \sigma \in \Sigma^f_{\text{st}}\}$.

The maximal model of T_f is a graph $G^f_{\max}$ obtained from chasing the saturated instance for f with the set of mappings. The complexity of constructing the maximal model for f is given by the following proposition:

Proposition 6.3.6. Let m be the maximum number of atoms in a st-tgd in Σ^f_{st}. Let l be the number of st-tgds in Σ^f_{st}. The construction of the maximal model for f is bounded by $l \cdot (n \cdot k + m) + l$ where n is the number of inclusion dependencies and k is the maximum arity of relational name in $\mathscr{R}$.

Proof. Let l be the number of st-tgds in Σ^f_{st}. We use the localized chase twice per st-tgd and we do repeat it l times obtaining the union of all this localized-chased canonical

instances. By Proposition 6.3.4 the result of applying each localized chase is bounded by $n \cdot k + m$. Thus, the saturated instance for f is bounded by $l \cdot (n \cdot k + m)$. Then we chase this result with Σ_{st}^{f}, which by Beeri and Vardi [Beeri & Vardi 1984], we obtain that the maximal model is bounded by $l \cdot (n \cdot k + m) + l$. □

6.3.2 Algorithm

Now, we present the M3 algorithm based on the use of minimal and maximal models. The input is a schema-less data exchange setting $\mathscr{E}_{\mathscr{A}} = (\mathbf{R}, \Sigma_{\text{st}}, \mathbf{F})$. The aim of the elicitation algorithm is to generate a shape definition for each type that covers all possible graphs produced by any instance of $\mathbf{R}$ with the mappings.

Algorithm 5: M3: Elicitation algorithm.

Input: a schema-less data exchange setting $\mathscr{E}_{\mathscr{A}} = (\mathbf{R}, \Sigma_{\text{st}}, \mathbf{F})$
Output: a shapes schema $\mathbf{S} = (\mathscr{T}, \delta)$.

$\mathscr{T} \coloneqq \{T_f \mid \exists \sigma \in \Sigma_{\text{st}}, \mathit{constructor}(\sigma) = f\}$;
for T_f *in* $\mathscr{T}$ **do**
  $\mathscr{G} \coloneqq \emptyset$;
  $I^{f}_{\text{max}} \coloneqq \emptyset$;
  for σ *in* Σ^{f}_{st} **do**
    $I_\sigma \coloneqq \{\mathit{body}(\sigma)\}$;
    $I^{*}_{\sigma} \coloneqq \ell\text{-}\mathit{chase}^{\Sigma_{\text{st}}}_{f}(I_\sigma, \Sigma_{\text{ind}})$;
    $I^{**}_{\sigma} \coloneqq \mathit{chase}(I^{*}_{\sigma}, \Sigma_{\text{fd}})$;
    $G^{\sigma}_{\text{min}} \coloneqq \mathit{chase}(I^{**}_{\sigma}, \Sigma^{f}_{\text{st}})$;
    add G^{σ}_{min} to $\mathscr{G}$;
    θ assigns to every non-share variable a fresh variable;
    $J_\sigma \coloneqq \{\mathit{body}(\theta(\sigma))\}$;
    $J^{*}_{\sigma} \coloneqq \ell\text{-}\mathit{chase}^{\Sigma_{\text{st}}}_{f}(J_\sigma, \Sigma_{\text{ind}})$;
    $I^{**} \coloneqq \mathit{chase}(I^{*}_{\sigma} \cup J^{*}_{\sigma}, \Sigma_{\text{fd}})$;
    $I^{f}_{\text{max}} \coloneqq I^{f}_{\text{max}} \cup \{I^{**}\}$;
  end
  $G^{f}_{\text{max}} \coloneqq \mathit{chase}(I^{f}_{\text{max}}, \Sigma^{f}_{\text{st}})$;
  add G^{f}_{max} to $\mathscr{G}$;
  $\delta(T_f) \coloneqq \mathit{fit}(\{\mathit{ShExpr}(G, f(x)) \mid G \in \mathscr{G}\})$;
end
return $\mathbf{S} = (\mathscr{T}, \delta)$;

Algorithm 5 begins with the definition of the set of types $\mathscr{T}$ obtained from the IRI constructors that appeared in the subject position of a st-tgd in Σ_{st}. For each type T_f in $\mathscr{T}$, we compute its shape definition as follows. We construct all the minimal models

and the maximal model. Then we fit the set of shape expressions obtained from each model, and assign the fitted shape expression to the shape definition of T_f.

6.4 Soundness

Now, we show that M3 is sound using the notion of embedding.

Theorem 6.4.1 (Soundness)**.** *Let* $\mathbf{S}$ *be a shapes schema obtained by M3. It holds that* $\mathbf{S}$ *is sound w.r.t.* $\mathscr{E}_d$.

Proof. Let $\mathbf{S}$ be the shapes schema obtained with M3. Take $G \in T(\mathscr{E}_d)$. We construct implicitly the typing of G by using the type of a node $n \in nodes(G)$ that corresponds to the IRI constructor used to construct n. We prove that $G \in L_{\mathsf{C}}(\mathbf{S})$. Let $G_{\mathbf{S}}$ be the shape graph of $\mathbf{S}$. By definition, a graph $G \models \mathbf{S}$ if there is an embedding R from G in $G_{\mathbf{S}}$. We construct a relation R and show it is an embedding. Let $R = \{(n, T) \in nodes(G) \times nodes(G_{\mathbf{S}}) \mid T \in typing(n)\}$. We take two nodes $n \in nodes(G)$ and $T \in nodes(G_{\mathbf{S}})$ and show there is a witness of simulation of n by T. Take $(n, p, m) \in out_G(n)$. By construction of shape expressions of minimal and maximal models, we know there is a triple constraint with the predicate label p and the type of the target node i.e., $p :: T'^{\mu} \in \delta(T)$ for some μ multiplicity and some T' the target type of m. Thus, $(T, (p, \mu), T') \in G_{\mathbf{S}}$. Then, we define the function $\lambda : out_G(n) \to out_{G_{\mathbf{S}}}(T)$. Because we have constructed implicitly the types of G, then for every outgoing edge of n there is an outgoing edge in $out_{G_{\mathbf{S}}}(T)$. Because $T' \in typing(m)$, $(m, T') \in R$.

Now, we take $(T, (q, \mu), T') \in out_{G_{\mathbf{S}}}(T)$ and show by contradiction that

$$\bigoplus \{occur_G((n, q, o)) \mid (n, q, o) \in out_G(n), \lambda((n, q, o)) = (T, (q, \mu), T')\} \subseteq occur_{G_{\mathbf{S}}}((T, (q, \mu), T').$$

So we have two possible cases one corresponding the lower bound and the other corresponding the upper bound

1. $\{0, \ldots\} \not\subseteq \{1, \ldots\}$
2. $\{1, 2, \ldots\} \not\subseteq \{1\}$

For the first case, we assume G has no outgoing edges from n with q and target type of the node being T' and assume the shape definition for T requires at least one outgoing edge with q to a node of type T'. By construction of the shape expression in M3, there is a st-tgd $\sigma \in \Sigma_{\text{st}}$ of the form

$$\varphi(\mathbf{u}, v) \Rightarrow \mathit{Triple}(f(\mathbf{u}), q, v)$$

that is triggered in some locally-chased canonical instance $I_{\sigma'}$ of some mapping σ'. By construction of $I_{\sigma'}$, there is set of inclusion dependencies Σ_{ind} that have been applied from the canonical source instance of σ' such that $\mathit{body}(\sigma)$ has been triggered. Because $G \in T(\mathscr{C}_{\mathscr{A}})$, there is an instance $I \in \mathit{Inst}(\mathbf{R})$ such that $I \cup G \models \Sigma_{\text{st}}$. Since $\sigma \in \Sigma_{\text{st}}$ and G does not have an outgoing edge from n with q, then there is no homomorphism $h(\mathit{body}(\sigma)) \subseteq I$. Because $\sigma' \in \Sigma_{\text{st}}$ and not restriction is given of σ', we assume that there is $h'(\mathit{body}(\sigma')) \subseteq I$. Since Σ_{ind} exist and the value flow dependency graph indicates that value from the position must be send to the marked position that appears in the body of σ, then I must satisfy Σ_{ind}. However, no homomorphism is in the body of σ. Thus, I does not satisfy Σ_{ind} and $I \notin \mathit{Inst}(\mathbf{R})$; a contradiction.

For the second case, we assume G has more than one outgoing edge from n with q and target type of the node being T' and assume the shape definition for T restricts to have only one outgoing edge with q to a node of type T'. By construction of the shape expression by M3, there is $\sigma \in \Sigma_{\text{st}}$ of the form $\varphi(\mathbf{x}, y) \Rightarrow \mathit{Triple}(f(\mathbf{x}), q, y)$ that is triggered in every minimal model and maximal model for T. Also, there is a functional dependency $\rho \in \Sigma_{\text{fd}}$ that is applied such that in the construction of the maximal model for T, some values in the saturated instance for T are equated and from these values one is used to generate a triple with q. This implies that there is some relation in $\varphi(\mathbf{x}, y)$ such that the attributes in positions of $\mathbf{x}$ implies y. Because $G \in T(\mathscr{C}_{\mathscr{A}})$, there is an instance $I \in \mathit{Inst}(\mathbf{R})$ such that $I \cup G \models \Sigma_{\text{st}}$. We apply backwards the mappings on G and have that for $\sigma \in \Sigma_{\text{st}}$ there must be different homomorphisms that assign variables of σ to constant values such that $h_1(y) \neq h_2(y) \neq \ldots$. Because there is a functional dependency $\rho \in \Sigma_{\text{fd}}$, on the attributes corresponding to $\mathbf{x}$, then those constant values will be equated. Then $I \notin \mathit{Inst}(\mathbf{R})$; a contradiction. □

6.5 Completeness

We identify a class of mappings where M3 algorithm is complete i.e., for any schema-less data exchange setting in such class, the constructed target schema is complete. This class is based on the classical translation of an entity-relationship diagram to mappings. We recall the notion of an entity-relationship diagram and present a natural translation to a schema-less data exchange setting with the following example.

Example 6.5.1. Consider ER diagram show in Figure 6.8. This diagram presents two entities $Project$ and $Supervisor$ and the relationship $manages$ that connects these two entities. The entity $Project$ has two attributes: id and $title$ where the attribute id is the key. The entity $Supervisor$ has three attributes: id, $name$ and $office$. The key attribute of this entity is id. The orientation of relationship $manages$, $\rightarrow$, indicates that we forbid a project to exist if there is no supervisor that organizes it and that for each project there is at most one supervisor that manages it. The orientation of relationship $oversees$ indicates for each project zero or more supervisors can oversee it and every supervisor can oversee zero or more projects. We use the classical translation of ER

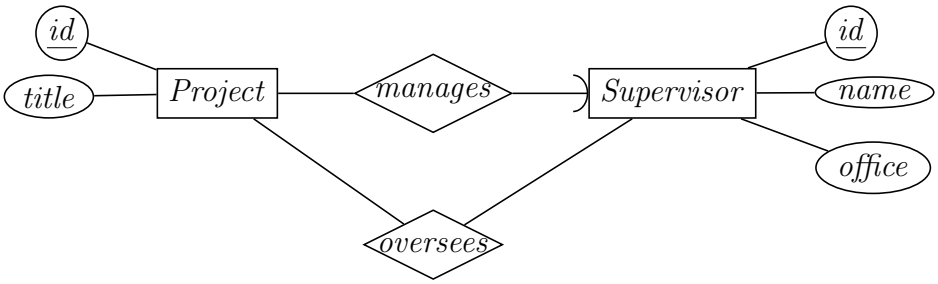


Figure 6.8: An example of ER diagram.

diagram to relational schema [Garcia-Molina *et al.* 2009]. For instance, for the entity $Project$, we create a relation with the attributes id and $title$. We create primary key in the relation with the key attribute of the entity $Project$. Because the entity $Project$ has a referential constraint to $Supervisor$, then we add the key of the entity $Supervisor$ as an attribute to the relation. Because of this relationship, we create the following inclusion dependency $Project[idSup] \subseteq Supervisor[id]$. We do the same for entity $Supervisor$. Next, we treat the relationship $oversees$. We create a relation with this name and use the primary keys of $Project$ and $Supervisor$ as the attributes of the relation $Oversees$. The primary key of this relation is all the attributes. We obtain the following

relational schema with three relations $Project(\underline{id}, title, idSup)$, $Supervisor(\underline{id}, name)$ and $Oversees(\underline{idProj, idSup})$ and the set of source dependencies (left fds and right inclusion dependencies).

$Project : id \rightarrow id\ title\ idSup$ $\quad$ $Project[idSup] \subseteq Supervisor[id]$

$Supervisor : id \rightarrow id\ name\ office$ $\quad$ $Oversees[idProj] \subseteq Project[id]$

$Oversees : idProj\ idSup \rightarrow idProj\ idSup$ $\quad$ $Oversees[idSup] \subseteq Supervisor[id]$

Now, from this ER diagram and assuming the relational schema is constructed as above, we construct a set of mappings as follows. For every entity, we identify the relation that corresponds to it, assign a unique IRI constructor and for every attribute related to the entity, we create a mapping with objects as literals. If the relation has any referential keys due to a relationship with orientation $\twoheadrightarrow$, then a mapping is created with object as IRI node where the IRI constructor is the one assigned to the entity and the argument is the key attributes. For instance, mapping (6.21) is created from relation with referential constraint. The subject in these mappings is composed of the IRI constructor assigned to the entity and the primary key of the entity. Next, for every relationship between E_1 and E_2 with orientations $-$ and $\twoheadrightarrow$, we create a mapping with subject composed of the IRI constructor assigned to E_1 and the argument is the key attributes of E_1, the predicate is the name of the relation, and the object composed of the IRI constructor assigned to E_2 and the argument of this object is the key attributes of E_2. For instance, mapping (6.25). We obtain the following mappings:

$$Project(id, title, idSup) \Rightarrow Triple(pro2iri(id), \mathsf{id}, id), \tag{6.19}$$

$$Project(id, title, idSup) \Rightarrow Triple(pro2iri(id), \mathsf{title}, title), \tag{6.20}$$

$$Project(id, title, idSup) \Rightarrow Triple(pro2iri(id), \mathsf{manages}, sup2iri(idSup)), \tag{6.21}$$

$$Supervisor(id, name, office) \Rightarrow Triple(sup2iri(id), \mathsf{id}, id), \tag{6.22}$$

$$Supervisor(id, name, office) \Rightarrow Triple(sup2iri(id), \mathsf{name}, name), \tag{6.23}$$

$$Supervisor(id, name, office) \Rightarrow Triple(sup2iri(id), \mathsf{office}, office), \tag{6.24}$$

$$Oversees(idPro, idSup) \Rightarrow Triple(pro2iri(idPro), \mathsf{oversees}, sup2iri(idSup)) \tag{6.25}$$

where $pro2iri$ and $sup2iri$ are the IRI constructors assigned to entities $Project$ and $Supervisor$.

The output of M3 with this schema-less data exchange setting composed of the relational schema obtained above, the set of mappings obtained from the ER diagram, and the IRI constructors assigned to the entities, is the following shapes schema.

$$T_{Proj} \rightarrow \mathsf{id} :: Lit^{1}; \mathsf{title} :: Lit^{1}; \mathsf{manages} :: T^{1}_{Sup}; \mathsf{oversees} :: T^{*}_{Sup}.$$
$$T_{Sup} \rightarrow \mathsf{id} :: Lit^{1}; \mathsf{name} :: Lit^{1}; \mathsf{office} :: Lit^{1}.$$

This shapes schema is a complete schema. □

Our claim holds for a subclass of ER diagrams, which we introduce next. An entity-relation (ER) diagram is a graphical language used to design databases. This language uses entities and relations. A *entity* is a class of objects of the same type. An object is described by a set of attributes, where a subset of them is used to identify the object within the class. A *relationship* connects two entities E_1 and E_2. We distinguish three types of relationship: many-to-many relationship, many-to-one relationship, and many-to-one with referential integrity constraint. The first type indicates that an object of E_1 can be related with many objects of E_2 and vice-versa. The second type indicates that two objects of E_2 cannot be related with the same object of E_1. The third type indicates that an object of E_1 must be related with an object of E_2 and at most one.

We assume an infinite set of names $\mathcal{N}$. We identify the set of entity names $Ent \subseteq \mathcal{N}$, the set of attribute names $\mathscr{A} \subseteq \mathcal{N}$ and the set of relation names $Rel \subseteq \mathcal{N}$. We assume that these sets are pairwise disjoint. Formally, a ***flat ER diagram*** is a tuple $ER = (Ent, Rel, src, tgt, attr, key, orient)$ where

- Ent is the set of entities, which are the set of nodes,
- Rel is the set of edges,
- $src : Rel \rightarrow Ent$ and $tgt : Rel \rightarrow Ent$ identify respectively, the origin entity of the relationship and the end entity of the relationship,
- $attr : Ent \rightarrow 2^{\mathscr{A}}$ assigns to an entity a finite set of attributes,

- the function $key : Ent \to 2^{\mathscr{A}}$ assigns to every entity $E \in Ent$ a finite set of attributes $A \subseteq attr(E)$ that is a key.

- $orient : Rel \to \{-, \twoheadrightarrow, \to\}$ specifies the type of a relation as follows. The symbol $-$ specifies the many-to-many relationship. The symbol $\twoheadrightarrow$ specifies the many-to-one relationship and the last symbol specifies the many-to-one relationship with referential integrity constraint.

By $rel\text{-}schema(ER)$, we denote the standard function of constructing a relational schema from a ER diagram presented in Example 6.5.1. Formally, $rel\text{-}schema(ER) = (\mathscr{R}, attrs, \Sigma_{\text{fd}}, \Sigma_{\text{ind}})$ where

- $\mathscr{R} = \{R_E \mid \exists E \in Ent.\,\} \cup \{R_r \mid \exists r \in Rel.\, orient(r) \in \{-, \twoheadrightarrow\}\}$.

- For every $E \in Ent$, there is $R_E \in \mathscr{R}$ such that

  - if there is no $r \in Rel$ such that $src(r) = E$ and $orient(r) = \to$, then $attrs(R_E) = attr(E)$.

  - if there exists $r \in Rel$ such that $src(r) = E$ and $orient(r) = \to$, then

$$attrs(R_E) = attr(E) \cup \bigcup \{key(E') \mid r \in Rel.\, orient(r) = \to, src(r) = E, tgt(r) = E'\}.$$

- the set of functional dependencies is

$$\begin{aligned}
\Sigma_{\text{fd}} = \{&R_E : A \to B \mid \exists E \in Ent.\ \nexists r \in Rel.\, src(r) = E, orient(r) = \to, \\
&A = key(E), B = attr(E)\} \cup \\
&\{R_E : A \to B \mid \exists E \in Ent. \forall r \in Rel.\, src(r) = E, orient(r) = \to, \\
&A = key(E), \\
&B = \bigcup \{key(E') \mid r \in Rel.\, orient(r) = \to, src(r) = E, tgt(r) = E'\}\} \cup \\
&\{R_r : A \to A \mid \exists r \in Rel.\, orient(r) = -, A = key(E_1) \cup key(E_2)\} \cup \\
&\{R_r : A \to B \mid \exists r \in Rel.\, orient(r) = \twoheadrightarrow, A = key(E_1), B = key(E_2)\}
\end{aligned}$$

- the set of inclusion dependencies is

$$\begin{aligned}\Sigma_{\text{ind}} = \{&R_r[A] \subseteq R_{E_1}[A], R_r[B] \subseteq R_{E_2}[B] \mid \exists r \in \mathit{Rel}.\, \mathit{orient}(r) \in \{-, \rightarrow\},\\ &\mathit{src}(r) = E_1, \mathit{tgt}(r) = E_2, A = \mathit{key}(E_1), B = \mathit{key}(E_2)\} \cup\\ &\{R_{E_1}[A] \subseteq R_{E_2}[A] \mid \exists r \in \mathit{Rel}.\, \mathit{orient}(r) = \rightarrow, \mathit{src}(r) = E_1, \mathit{tgt}(r) = E_2,\\ &A = \mathit{key}(E_2)\}\end{aligned}$$

We introduce a function, denoted by $\mathit{mapping}$ that takes an ER diagram and returns a schema mapping from relational schema to graphs. By $\mathit{mapping}(ER)$, we denote the set of mappings constructed from an ER diagram. In this function, the set of IRI constructors is defined as $\mathscr{F} = \{f_E \mid E \in \mathit{Ent}\}$. First, we construct its relational schema and the construction of the set of mappings is as follows. For every $E \in \mathit{Ent}$, we assign a unique IRI constructor f_E and let $R_E \in \mathscr{R}$ be the corresponding relation to E of the form

$$R_E(\mathbf{x}, a_1, \ldots, a_n, \mathbf{y}_1, \ldots, \mathbf{y}_k)$$

where

- $\mathbf{x}$ corresponds to the primary key,
- $a_1, \ldots, a_n$ to the attributes of the entity,
- each $\mathbf{y}_1, \ldots, \mathbf{y}_k$ corresponds to each many-to-one relationship with referential integrity constraints of E i.e., there is an inclusion dependency $R_E[\mathbf{y_l}] \subseteq R_{E_l}[\mathbf{x}']$ where $l \in \{1, \ldots, k\}$ with k being the number of many-to-one relationship with referential integrity constraints, and
- $\mathbf{x}'$ corresponds to the primary key of R_{E_l}.

For each corresponding many-to-one relationship with referential integrity constraint, we denote as q_l where $l \in \{1, \ldots, k\}$. Then, we create the following mappings for E:

$$\begin{aligned}R_E(\mathbf{x}, a_1, \ldots, a_n, \mathbf{y}_1, \ldots, \mathbf{y}_k) &\Rightarrow \mathit{Triple}(f_E(\mathbf{x}), a_i, a_i)\\ R_E(\mathbf{x}, a_1, \ldots, a_n, \mathbf{y}_1, \ldots, \mathbf{y}_k) &\Rightarrow \mathit{Triple}(f_E(\mathbf{x}), q_l, f_{E_i}(\mathbf{y}_l))\end{aligned}$$

where $i \in \{1, \ldots, n\}$.

For every many-to-many relationship and many-to-one relationship r between two entities E_1 and E_2, there is a relation of the form $R_r(\mathbf{y}, \mathbf{z})$ where $\mathbf{y}$ and $\mathbf{z}$ correspond to the primary key of E_1 and E_2, respectively. Moreover, there are two inclusion dependencies $R_r[\mathbf{y}] \subseteq R_{E_1}[\mathbf{y}]$ and $R_r[\mathbf{z}] \subseteq R_{E_2}[\mathbf{z}']$. Then, we create the following mapping

$$R_r(\mathbf{y}, \mathbf{z}) \Rightarrow \mathit{Triple}(f_{E_1}(\mathbf{y}), r, f_{E_2}(\mathbf{z})),$$

where f_{E_1} and f_{E_2} are two IRI constructors assigned to E_1 and E_2, respectively.

Interestingly, from a given ER diagram, we can produce a shapes schema for graphs representing the information that is modeled by the ER diagram. This is a natural construction. By, $\mathit{sh\text{-}schema}(ER)$ we denote the function of constructing a shapes schema as follows. For every entity $E \in \mathit{Ent}$, we assign a distinguish type T_E and for every attribute of E, we create a triple constraint where the attribute name is the property label and the target type is Lit. For every relationship of E that relates some other entity $E' \in \mathit{Ent}$, we create a triple constraint where the property label is the name of the relationship and the target type is E'. The multiplicity of each triple constraint where the target type is not a literal is assigned according to the type of relationship.

- If the relationship is many-to-one with referential integrity constraint, then the multiplicity is `1` because this relationship allows an object of E to be related with at most one object of E' and since the object is identified by some key, then is one to one object.

- If relationship is many-to-many, then the multiplicity is $*$ because this relationship allows an object of E to be related with zero or more objects of E'.

- If relationship is many-to-one, then the multiplicity is `?` because this relationship does not allow two objects of E to be related with the same object of E' i.e, a object of E is related with zero or one object of E'.

Otherwise, if the target is Lit, then the multiplicity is `1`. Formally, $\mathit{sh\text{-}schema}(ER) = (\mathscr{T}, \delta)$ where

- the set of types is $\mathscr{T} = \{T_E \mid E \in \mathit{Ent}\}$,

- for every type T_E its shape definition is

$$\begin{aligned}\delta(T_E) = \{a :: Lit \mid a \in attr(E)\} \cup \\ \{r :: T^{\mathtt{1}}_{E'} \mid r \in Rel, src(r) = E, tgt(r) = E', orient(r) = -\} \cup \\ \{r :: T^{*}_{E'} \mid r \in Rel, src(r) = E, tgt(r) = E', orient(r) = \twoheadrightarrow\} \cup \\ \{r :: T^{?}_{E'} \mid r \in Rel, src(r) = E, tgt(r) = E', orient(r) = \rightarrow\}\end{aligned}$$

Now, we claim the following theorem.

Theorem 6.5.2. *For any flat ER diagram ER, let $\mathbf{F}$ be defined in $mapping(ER)$ and let $\mathscr{E}^{\mathrm{ER}}_{d} = (rel\text{-}schema(ER), mapping(ER), \mathbf{F})$, it holds that*

ER.1 *M3 on $\mathscr{E}^{\mathrm{ER}}_{d}$ returns $sh\text{-}schema(ER)$,*

ER.2 *$sh\text{-}schema(ER)$ is a sound and complete shapes schema for $\mathscr{E}^{\mathrm{ER}}_{d}$.*

Proof. Take a flat ER diagram ER. The proof of ER.1 follows from construction of the algorithm and definition of $sh\text{-}schema(ER)$ i.e., all the predicates in Σ_{st} come from relationships and attributes that are mapped to property labels as done in $sh\text{-}schema(ER)$ and each triple constraint obtained by M3 is in $sh\text{-}schema(ER)$ and the multiplicities are the same.

Now, we prove ER.2 by proving that M3 is sound and complete for $\mathscr{E}^{\mathrm{ER}}_{d}$. By Theorem 6.4.1 and because $\mathscr{E}^{\mathrm{ER}}_{d}$ is a subclass of a schema-less data exchange setting, M3 is sound for $\mathscr{E}^{\mathrm{ER}}_{d}$.

Now, we prove completeness of M3. Let $\mathbf{S} = sh\text{-}schema(ER)$. Take a graph $G \in L_{\mathsf{C}}(\mathbf{S})$. Assume G uses IRI nodes of the form $f(k_1, \ldots, k_n)$ for some $f \in \mathscr{F}$, $k_i \in \mathsf{Const}$ where $i \in \{1, \ldots, n\}$ and $n = |key(E)|$ for some $E \in Ent$ and f corresponds to E. We apply backwards the mappings on G obtaining an instance I. The form of $mapping(ER)$ and $\mathbf{S}$ ensures that the backchase step is well-defined as follows. If we take a node $n \in nodes(G)$ then every outgoing edge label of n is either attribute or relationship name, other kind of label is not accepted by $\mathbf{S}$. Since every attribute and relationship name is considered in $rel\text{-}schema(ER)$, then for every outgoing edge of n, there is tuple in a relation with the values of the source and target node. For

instance, $Triple(f(1), a, 3)$, we have a fact $R(1, \ldots, 3, \ldots)$. Also, no two st-tgds obtained by $mapping(ER)$ have same subject and predicate and every argument of an IRI constructor is a primary key in the relation that appears in the body of a st-tgd.

We prove by contradiction that I satisfies the set of dependencies of $\mathbf{R}$. Assume that $I \not\models \Sigma_{\mathbf{R}}$. Then, there are two possible cases.

- Not satisfying key constraints. It implies that

  1. we have $\{R(d, a_1, \ldots, a_m), R(d, b_1, \ldots, b_m)\} \subseteq I$ for some $a_1, \ldots, a_m \in \mathsf{Const}$, $b_1, \ldots, b_m \in \mathsf{Const}$ and $d \in \mathsf{Const}$ such that for every $i \in \{1, \ldots, m\}$, it holds that $a_i \neq b_i$, and
  2. there is $R(x, a_1, \ldots, a_m, \mathbf{y}_1, \ldots, \mathbf{y_k})$ in the relational schema where x is the key for R.

  Because x is a key in R and by definition of $mapping(ER)$, x is used in some IRI constructor $f \in \mathscr{F}$, then M3 obtains for T_f a triple constraint $a :: Lit^1$ for some $a \in attrs(R)$. From (1) and $mapping(ER)$, there is mapping with IRI constructor f assigned to R and predicate a in its triple atom. Because I is obtained by applying backwards the mappings, G contains at least the triples $\{(f(d), a, a_k), (f(d), a, b_k)\}$ for some $k \in \{1, \ldots, m\}$. Since $G \in L_{\mathsf{C}}(\mathbf{S})$ and the shape definition for T_f has $a :: Lit^1$, then $G \not\models \mathbf{S}$; a contradiction.

- Not satisfying a foreign key. It implies that for a fact in I such as the following $R(d, a_1, \ldots, a_n, \mathbf{b}_1, \ldots, \mathbf{b}_k) \subseteq I$, there is no fact with $P(\mathbf{b}_l, c_1, \ldots, c_m) \subseteq I$ for some $d \in \mathsf{Const}$, some vector of constants $\mathbf{b}_l$ and some constants $c_1, \ldots, c_m$ where $l \in \{1, \ldots, k\}$ and there is an inclusion dependency $R[\mathbf{b}_k] \subseteq P[\mathbf{b}_k]$. We choose $l \in \{1, \ldots, k\}$ and $u \in \{1, \ldots, m\}$, and by $mapping(ER)$, we identify two mappings

$$R(x, a_1, \ldots, a_n, \mathbf{y}_1, \ldots, \mathbf{y}_k) \Rightarrow Triple(f(x), q_l, g(\mathbf{y}_l))$$
$$P(\mathbf{x}', z_1, \ldots, z_m) \Rightarrow Triple(g(\mathbf{x}'), z_u, z_u)$$

  where $f, g \in \mathscr{F}$ and f is assigned to R, and g is assigned to P. M3 obtains for T_g a triple constraint $z_u :: Lit^1$. Because I is obtained by applying backwards the

mappings, then G contains triple $Triple(f(d), q_l, g(\mathbf{b}_l))$ and does not contain an outgoing edge with z_u from $g(\mathbf{b}_l)$; a contradiction because the shape definition for T_g has $z_u :: Lit^{\mathtt{1}}$ and $G \in L_{\mathsf{C}}(\mathbf{S})$.

Now, let $chase(I, \Sigma_{\text{st}}) = G'$. We prove by contradiction that $G' \cong G$. For the $\Rightarrow$ direction, take a node $n \in nodes(G')$ and assume n not in G'. Since every node is produced by the application of an IRI constructor then let be f the IRI constructor. The assumption only is possible if $f(x) = n$ and $f(x) = m$ being m in G'; a contradiction because the IRI constructor only generates precisely one IRI value. For the $\Leftarrow$ direction, take a triple $(n, a, m) \in G$ and assume this triple is not in G'. By construction of shape expressions of minimal and maximal models and $G \in L_{\mathsf{C}}(\mathbf{S})$, then there is a triple constraint with $a :: T^{\mu}$ for some $T \in \mathscr{T}$ and some $\mu \in \{\mathtt{1}, \mathtt{?}, \mathtt{+}, \mathtt{*}\}$ and there is a mapping σ with predicate a in the head atom. Since we applied backwards the mappings to obtain I, we know there is a homomorphism h such that $h(body(\sigma)) \subseteq I$. We chase I with the mappings and we obtain that (n, a, m) is in G'; a contradiction. □

Given a flat ER diagram ER, we introduce a class of relational schemas that characterizes $rel\text{-}schema(ER)$. A relational schema $\mathbf{R} = (\mathscr{R}, attrs, \Sigma_{\text{fd}}, \Sigma_{\text{ind}})$ is flat if for every $R \in \mathscr{R}$, it holds that

- for every fd of the form $R : A \to B$ in Σ_{fd} for some $A \subseteq attrs(R)$, it holds that $B = attrs(R)$,
- for every inclusion dependency of the form $R[A] \subseteq P[B]$ for some $P \in \mathscr{R}$, $A \subseteq attrs(R), B \subseteq attrs(P)$, we have that
  - B is the primary key of P,
  - if $A \subseteq K$ where K is the primary key of R, then there is a relation $Q \in \mathscr{R}$ such that the set $E = attrs(R) \setminus A$ has an inclusion dependency of the form $R[E] \subseteq Q[E']$ for some $E' \subseteq attrs(Q)$,
  - there is no inclusion dependency of the form $P[B] \subseteq R[A]$.

A schema-less data exchange setting is *flat* if the relational schema is *flat*, and the set of mappings has a single body atom, the subject is used as key in the relation and there are no two st-tgds with the same subject and predicate. It is easy to see that the setting

$\mathscr{C}_d^{\mathrm{ER}} = (\mathit{rel\text{-}schema}(ER), \mathit{mapping}(ER), \mathbf{F})$ where $\mathbf{F}$ is defined in $\mathit{mapping}(ER)$ is flat.

The following theorem follows from Theorem 6.5.2.

Theorem 6.5.3. *M3 is sound and complete elicitation algorithm for every flat schema-less data exchange setting obtained from flat ER diagram.*

6.6 Negative results

It is possible to conceive an algorithm that is complete for a larger class of schema-less data exchange settings. However, in this section, we present two results that identify the inherent limitations of the task of producing a sound and complete schema. First, we show that a sound and complete output may be exponential.

Theorem 6.6.1. *For any $n \in \mathbb{N}$, there exists a relational schema and set of mappings of size polynomial in n such that the complete target schema has at least $2^n - 1$ types.*

Proof. For $n \in \mathbb{N}$, we construct a relational schema $\mathbf{R}_n$ with $n+1$ relations $\mathscr{R} = \{R(\underline{x}), P_1(\underline{x}, y_1, y_2), \ldots, P_n(\underline{x}, y_1, y_2)\}$ with the indicated key constraint and n inclusion dependencies $P_i[x] \subseteq R[x]$ for $i \in \{1, \ldots, n\}$. The set of mappings Σ_{st} consists of

$$R(x) \wedge P_i(x, y_1, y_2) \Rightarrow \mathit{Triple}(f(x), p_i, y_1) \wedge \mathit{Triple}(f(x), q_i, y_2)$$

for $i \in \{1, \ldots, n\}$ and some $f \in \mathscr{F}$.

Suppose we have a sound and complete schema $\mathbf{S}$ and assume by contradiction that $\mathbf{S}$ has less than $2^n - 1$ types. We take a enumeration $(X_1, \ldots, X_{2^n-1})$ of the collection $\mathscr{P}(\{1, \ldots, n\}) \setminus \{\emptyset\}$ of non-empty subsets of $\{1, \ldots, n\}$. Then, we construct an instance as follows:

$$I = \{R(i) \mid 0 \leq i \leq 2^n - 1\} \cup \{P_j(i, a, b) \mid 1 \leq j \leq n, 1 \leq i \leq 2^n - 1, j \in X_i\}.$$

Applying the mappings, we obtain a graph G with 2^n IRI nodes $n_1, \ldots, n_{2^n}$ such that $n_i = f(i)$ for $i \in \{1, \ldots, 2^n - 1\}$.

Now, we type the set of nodes of G against $\mathbf{S}$. Because $\mathbf{S}$ has less than $2^n - 1$ types, there are two nodes n_i and n_j that has the same type T where $i \neq j$ for some

$i, j \in \{1, \ldots, 2^n - 1\}$. We know $X_i \div X_j \neq \emptyset$ because $i \neq j$. We observe that for $l \in X_i \cap X_j$, both n_i and n_j have outgoing edges labeled with both p_l and q_l. Therefore, the type T has in its shape definition for any $l \in X_i \cap X_j$, $p_l :: Lit^{\mu}$ for some $\mu \in \{+, ?\}$ and $q_l :: Lit^{\mu'}$ for some $\mu' \in \{+, ?\}$. On the other hand, we observe that for any $l \in X_i \div X_j$, one of the nodes n_i or n_j has two outgoing edges p_l, q_l while the other has none. Therefore, the type T has in its shape definition for any $l \in X_i \div X_j$, $p_l :: Lit^{\mu}$ for some $\mu \in \{?, *\}$ and $q_l :: Lit^{\mu'}$ for some $\mu' \in \{?, *\}$.

Since $X_i \div X_j \neq \emptyset$, we choose $l \in X_i \div X_j$ and construct a graph G' such that there is a node $m \in nodes(G')$ that has the common parts from nodes $n_i, n_j \in nodes(G)$ and one outgoing edge with p_l and no outgoing edge with q_l. More precisely, we obtain

$$G' = \{\mathit{Triple}(m, p_k, a) \mid k \in X_i \cap X_j\} \cup \{\mathit{Triple}(m, q_k, b) \mid k \in X_i \cap X_j\} \cup \{\mathit{Triple}(m, p_l, a)\}.$$

We observe that m has type T and, therefore, $G' \in L_{\mathsf{C}}(\mathbf{S})$. Since the shapes schema is complete, then there must exist an instance such that application of Σ_{st} produces an isomorphic graph to G'. We evaluate the rules that can be triggered to produce G' and must contain at least the following set of facts.

$$\{R(c)\} \cup \{P_k(c, a, b) \mid k \in X_i \cap X_j\} \cup \{P_l(c, a, \bot)\},$$

for some $c \in \mathsf{Const}$. We observe that the set of facts $\{R(c), P_l(c, a, \bot)\}$ triggers the rule

$$R(x) \wedge P_l(x, y_1, y_2) \Rightarrow \mathit{Triple}(f(x), q_l, y_2)$$

causing that m has an outgoing edge from with label q_l i.e., when applying the mappings we obtain a graph G'' different from G'. Contradiction, the shapes schema needs to have at least $2^n - 1$ types. □

Next, we show that a sound and complete schema may not exist.

Theorem 6.6.2. *There is a schema-less data exchange setting $\mathscr{E}_d$ that does not admit a complete target schema w.r.t. $\mathscr{E}_d$.*

Proof. Let the relational schema be presented in Figure 6.9 and the set of mappings Σ_{st}

be as follows.

$$R(x, y) \Rightarrow \mathit{Triple}(f(x), r, f(y)) \tag{6.26}$$

$$P(x, y) \Rightarrow \mathit{Triple}(f(x), q, g(y)) \tag{6.27}$$

$$P(x, y) \wedge R(x, z) \Rightarrow \mathit{Triple}(g(y), p, f(z)) \tag{6.28}$$

Figure 6.9 indicates the following inclusion dependencies:

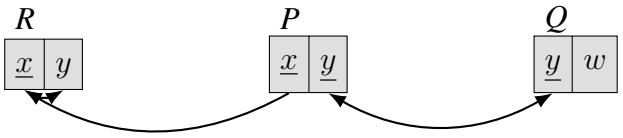


Figure 6.9: A relational schema.

$$R[x] \subseteq R[y] \qquad R[y] \subseteq R[x]$$

$$Q[x] \subseteq P[y] \qquad P[y] \subseteq Q[x]$$

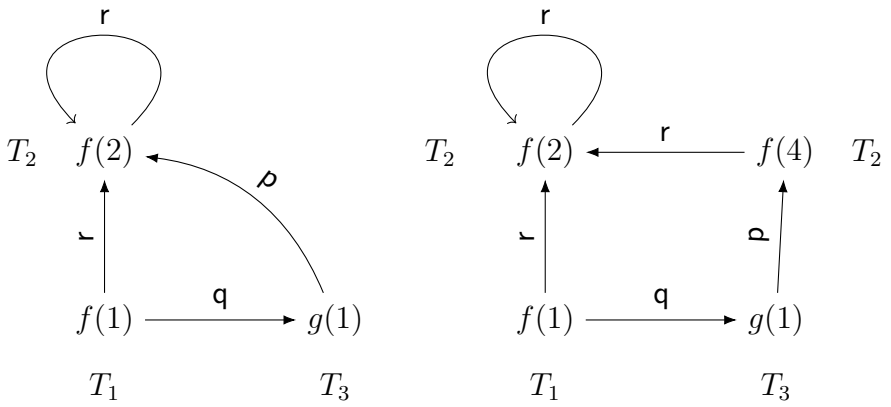
$$P[x] \subseteq R[x]$$

We prove by contradiction. Assume there is a complete shapes schema $\mathbf{S}$ such that a graph recognized by the language can have the triangle pattern. Let

$$I_0 = \{R(2,2), R(1,2), P(1,1), Q(1,3)\}$$

be an instance of the relational schema. The graph produced by the mappings is presented in Figure 6.10a. Since the graph satisfies the schema, then we assume that there are the types T_1, T_2 and T_3 such that T_1 is assigned to $f(1)$, T_2 is assigned to $f(2)$ and T_3 is assigned to $g(1)$. We observe the triangle pattern present in this graph. We break this pattern by cloning the node $f(2)$ such that for every outgoing edge of $f(2)$, we add the same outgoing edge in the cloned node having r-edge that goes from the cloned node to $f(2)$. Let this cloned node be $f(4)$. Then, we redirect the p-edge from $g(1)$ to be $f(4)$. Then, we assign the type T_2 to $f(4)$. Thus, we obtain a graph G' shown in Figure 6.10b. We observe that G' satisfies the schema because G' is bisimilar to G. In particular, $f(4)$ must satisfy T_2. Take the instance that constructs an isomorphic graph

to the one presented in Figure 6.10b. For edge from $f(1)$ to $f(2)$, we obtain the fact $R(1,2)$ and for edge $f(1)$ to $g(1)$, we obtain the facts $P(1,1)$. But, we observe that this set of facts $\{R(1,2), P(1,1)\}$ triggers rule (6.28) adding new edge from $g(1)$ to $f(2)$ to G' obtaining the triangle pattern. Contradiction because the graph obtained from the application of the mapping to the instance chosen is different from G'.



(a) Graph G produced by Σ_{st} on I_0 (b) Graph G' obtained from breaking pattern

Figure 6.10: Graphs recognized by $L_{\mathsf{C}}(\mathbf{S})$.

□

6.7 Conclusion

We have investigated the problem of shapes schema elicitation that aims at constructing a target schema for a schema-less relational to RDF data exchange setting, which is composed of the relational schema, set of mappings from relational schema to RDF graph and library of IRI constructors. This is a practically important problem in the context of the Web, because many RDF graphs on the Web have been exported from relational databases as evidenced by the proliferation of languages for mapping relational databases to RDF.

We have identified two desirable properties of the target shapes schema, which are soundness and completeness. The property of completeness of the target schema ensures that no trivial solutions are presented for this problem. Our solution is the M3 algorithm based on minimal and maximal models, which produces a sound schema for every schema-less data exchange setting. We have identified a practical class of

schema-less data exchange setting that follows from relational databases modeled by flat ER diagrams where M3 obtains a complete schema. Finally, we have shown that the task of producing sound and complete schemas has inherent limitations: a complete schema may be of exponential size and even the possibility of non existence of complete schema.

6.8 Related work

Schema elicitation can be generalized as constructing a schema from a set of mappings between two models and an input schema. We distinguish two kind of models: structured and semi-structured. By structured models, we mean relational databases where the schema is fixed i.e., if we add data to a relational database, we need to specify the schema. By semi-structured models, we mean XML, JSON, RDF databases and property graphs where we can have a database without specifying the schema. The problem of schema elicitation is trivial when both models are structured because the target schema is part of the description and together with the mappings contains all the information for the construction of the schema. The schema elicitation is relevant when the target schema is semi-structured.

In the case of mappings from structured model to RDF, to our knowledge there are no works that have tackled this problem. Our work is the first one in treating this case. In the case of mappings between XML databases, Groppe and Groppe [Groppe & Groppe 2008] has consider a variant of this problem because they do not have an input schema but a set a set of XML documents and the set of mappings is in a XSLT stylesheet, and the problem aims at constructing a XML Schema [Thompson *et al.* 2012] that characterizes the output language. Authors propose an algorithm that analyzes the XSLT stylesheet where the target schema obtained follows from the mappings. In their algorithm, there is a merging process and they notice, however, that there is a loss of information while during the merging. Thus, they do not consider the problem of completeness and most importantly they do not have an input schema.

Another related problem to schema elicitation is the typechecking problem that consists of evaluating if the target schema is sound w.r.t. an input schema and a set of mappings. Noga et al. [Alon *et al.* 2003] has studied decidable cases where a DTD is

sound w.r.t. a relational schema and set of mappings. In our approach, we construct a complete schema and we can see the typechecking as testing the containment between the target schema provided and the one that we obtained. Also, the schema inference problem is related with schema elicitation and it consists of constructing a schema from a set of instances of a model. For instance, Baazizi et al. [Baazizi *et al.* 2019] infer a schema from JSON databases, Li et al. [Li *et al.* 2019] infer a XML schema from a set of XML documents and Kellou-Menouer and Kedad [Kellou-Menouer & Kedad 2015] infer a RDF schema from a set of RDF databases. However, we do no present them because the setting does not consider mappings.

Conclusion

In this manuscript, we have investigated data exchange from relational databases to RDF graphs with target shapes schemas. We have presented different problems of interest in the data exchange problem due to the consideration of target schemas and use of declarative formalisms for expressing mappings such as R2RML. To study those problems we have formalized the constructive relational to RDF data exchange setting that captures a large fragment of R2RML.

For the error-prone process of designing mappings and consistency checking problem, we have developed a static analysis tool that allows to find non trivial modeling errors and illustrates them by constructing counter-examples, whose study allows to repair the modeling errors done in the mappings. Moreover, our techniques could potentially be used to manage complex process by decomposing the process into independent components. The decomposition process allows to lower the complexity and mitigate this error-prone process. One method to know if decomposition is possible is by testing independency of rules for which the technique for testing node kind consistency can be adapted to know if certain triples will appears in both results. Also, we have shown that checking consistency for a constructive setting is coNP-complete. A future work is to study redundancy of mappings by identifying mapping rules whose subtraction will not change anything. However, this problem requires additional techniques that we do not have study because redundancy is a a problem of containment and our techniques are not enough to solve this problem.

For the problem of certain query answering, we have proposed the notion of universal simulation solution. This solution has the property of a universal solution because certain answers for queries that are robust under simulation can be computed. Contrasted to other approaches, a universal simulation solution can be constructed for any

kind of constructive setting. Our contribution of this notion might give the foundation to know if a solution is good by showing that a solution must preserve the exchanged information and contain the missing information required by the schema as encoding of those constraints that are not satisfied. A challenge related to this problem is the construction of a solution in low complexity. We can construct a minimal-size universal simulation solution which is exponential in the size of the schema. Compared to other approaches, we materialize a solution, which is important in the context of data exchange, and constructs our good solution in low complexity. We have shown that the data complexity of computing certain answers in a universal simulation solution for forward NRE and any constructive setting is PTIME. Our approach does not follow from existing results on the standard relational data exchange, which are too limited in their expressive power. As a future work, we plan to study computing certain answers with full NRE-based queries and extending to non-Boolean queries.

For the problem of visual mapping language, we have proposed a visual mapping language (VML) to overcome the barriers of non-expert users in the specification of mappings. VML uses simple visual representations such arrows, boxes and lines, which combined make the mapping accessible and of easy use. However, the expressivity of VML is constrained in the tool to ensure the computation of a solution for a data exchange setting. We have developed ShERML that uses this language to specify mappings and performs the relational to RDF data exchange. This tool is an answer to the challenge of providing an interface that covers at most as possible the specification of constructive mappings. As a future work, we plan to extend the capabilities of ShERML so users can specify mappings with R2VML.

Finally for the problem of schema elicitation, we have defined a non-trivial algorithm that constructs a shapes schema using a method of minimal and maximal models. We can use this method to measure the quality of other algorithms. For instance, any other solution that does not recognize a minimal model is not correct, also a solution that recognizes graphs with more edges than the identified for the maximal model is a trivial solution. We have identified two properties that makes a target shapes schema a good schema. This two properties are soundness and completeness. Our algorithm is sound for every input and complete for a particular class of inputs derived from flat ER diagrams. As a future work, we plan to identify the precise class of inputs for which our

algorithm is complete. Our study also shows that inherent limitations of constructing target shapes schemas.

Our investigation has been focused on deterministic shapes schema and constructive st-tgds. It might be interesting to study each problem of interest in a data exchange setting with non-deterministic shapes schema and non-constructive st-tgds.

Bibliography

[Alon *et al.* 2003] Noga Alon, Tova Milo, Frank Neven, Dan Suciu and Victor Vianu. *Typechecking XML views of relational databases*. ACM Trans. Comput. Log., vol. 4, no. 3, pages 315–354, 2003.

[Amer-Yahia *et al.* 2002] Sihem Amer-Yahia, SungRan Cho, Laks V. S. Lakshmanan and Divesh Srivastava. *Tree pattern query minimization*. VLDB J., vol. 11, no. 4, pages 315–331, 2002.

[Arenas & Libkin 2008] Marcelo Arenas and Leonid Libkin. *XML data exchange: Consistency and query answering*. J. ACM, vol. 55, no. 2, pages 7:1–7:72, 2008.

[Arenas *et al.* 2010] Marcelo Arenas, Pablo Barceló, Leonid Libkin and Filip Murlak. Relational and XML data exchange. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

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

[Arenas *et al.* 2013] Marcelo Arenas, Pablo Barceló, Ronald Fagin and Leonid Libkin. *Solutions and query rewriting in data exchange*. Inf. Comput., vol. 228, pages 28–61, 2013.

[Auer *et al.* 2010] Sören Auer, Lee Feigenbaum, Daniel Miranker, Angela Fogarolli and Juan Sequeda. *Use Cases and Requirements for Mapping Relational Databases to RDF*, 2010.

[Baazizi *et al.* 2019] Amine Baazizi, Dario Colazzo, Giorgio Ghelli and Carlo Sartiani. *Parametric schema inference for massive JSON datasets*. VLDB J., vol. 28, no. 4, pages 497–521, 2019.

[Baget *et al.* 2011] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier and Eric Salvat. *On rules with existential variables: Walking the decidability line*. Artif. Intell., vol. 175, no. 9-10, pages 1620–1654, 2011.

[Bak *et al.* 2017] Jaroslaw Bak, Michal Blinkiewicz and Agnieszka Lawrynowicz. *User-friendly Visual Creation of R2RML Mappings in SQuaRE*. In Valentina Ivanova, Patrick Lambrix, Steffen Lohmann and Catia Pesquita, editors, Proceedings of the Third International Workshop on Visualization and Interaction for Ontologies and Linked Data co-located with the 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 22, 2017, volume 1947 of *CEUR Workshop Proceedings*, pages 139–150. CEUR-WS.org, 2017.

[Baker *et al.* 2012] Thomas Baker, Natasha Noy, Ralph Swick and Ivan Herman. *Semantic Web Case Studies and Use Cases*, 2012.

[Barceló *et al.* 2011] Pablo Barceló, Leonid Libkin and Juan L. Reutter. *Querying graph patterns*. In Maurizio Lenzerini and Thomas Schwentick, editors, Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece, pages 199–210. ACM, 2011.

[Barceló *et al.* 2013] Pablo Barceló, Jorge Pérez and Juan L. Reutter. *Schema mappings and data exchange for graph databases*. In Wang-Chiew Tan, Giovanna Guerrini, Barbara Catania and Anastasios Gounaris, editors, Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013, pages 189–200. ACM, 2013.

[Barceló 2009] Pablo Barceló. *Logical foundations of relational data exchange*. SIGMOD Rec., vol. 38, no. 1, pages 49–58, 2009.

[Beeri & Vardi 1984] Catriel Beeri and Moshe Y. Vardi. *A Proof Procedure for Data Dependencies*. J. ACM, vol. 31, no. 4, pages 718–741, 1984.

[Ben-Kiki *et al.* 2009] Oren Ben-Kiki, Clark Evans and Brian Ingerson. *YAML Ain't Markup Language (YAML)*, 9 2009.

[Benedikt *et al.* 2005] Michael Benedikt, Wenfei Fan and Gabriel M. Kuper. *Structural properties of XPath fragments*. Theor. Comput. Sci., vol. 336, no. 1, pages 3–31, 2005.

[Benedikt *et al.* 2017] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro and Efthymia Tsamoura. *Benchmarking the Chase*. In Emanuel Sallinger, Jan Van den Bussche and Floris Geerts, editors, Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017, pages 37–52. ACM, 2017.

[Bienvenu *et al.* 2013] Meghyn Bienvenu, Magdalena Ortiz and Mantas Simkus. *Conjunctive Regular Path Queries in Lightweight Description Logics*. In Francesca Rossi, editor, IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, pages 761–767. IJCAI/AAAI, 2013.

[Bojańczyk *et al.* 2013] Mikołaj Bojańczyk, Leszek A. Kołodziejczyk and Filip Murlak. *Solutions in XML data exchange*. Journal of Computer and System Sciences, vol. 79, no. 6, pages 785 – 815, 2013. JCSS Foundations of Data Management.

[Boneva *et al.* 2015] Iovka Boneva, Angela Bonifati and Radu Ciucanu. *Graph Data Exchange with Target Constraints*. In Peter M. Fischer, Gustavo Alonso, Marcelo Arenas and Floris Geerts, editors, Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), Brussels, Belgium, March 27th, 2015, volume 1330 of *CEUR Workshop Proceedings*, pages 171–176. CEUR-WS.org, 2015.

[Boneva *et al.* 2017] Iovka Boneva, José Emilio Labra Gayo and Eric G. Prud'hommeaux. *Semantics and Validation of Shapes Schemas for RDF*. In Claudia d'Amato, Miriam Fernández, Valentina A. M. Tamma, Freddy

Lécué, Philippe Cudré-Mauroux, Juan F. Sequeda, Christoph Lange and Jeff Heflin, editors, The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I, volume 10587 of *Lecture Notes in Computer Science*, pages 104–120. Springer, 2017.

[Boneva *et al.* 2018] Iovka Boneva, Jose Lozano and Slawomir Staworko. *Relational to RDF Data Exchange in Presence of a Shape Expression Schema*. In Dan Olteanu and Barbara Poblete, editors, Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, May 21-25, 2018, volume 2100 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.

[Boneva *et al.* 2019] Iovka Boneva, Jose Martin Lozano Aparicio and Slawomir Staworko. *ShERML: Mapping Relational Data to RDF*. In Mari Carmen Suárez-Figueroa, Gong Cheng, Anna Lisa Gentile, Christophe Guéret, C. Maria Keet and Abraham Bernstein, editors, Proceedings of the ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas) co-located with 18th International Semantic Web Conference (ISWC 2019), Auckland, New Zealand, October 26-30, 2019, volume 2456 of *CEUR Workshop Proceedings*, pages 213–216. CEUR-WS.org, 2019.

[Boneva *et al.* 2020] Iovka Boneva, Jose Lozano and Slawek Staworko. *Consistency and Certain Answers in Relational to RDF Data Exchange with Shape Constraints*. CoRR, 2020.

[Bray *et al.* 2008] Tim Bray, Jean Paoli, Eve Maler and C.M Sperberg-McQueen. *XML 1.0*. `https://www.w3.org/TR/xml/`, 2008.

[Calì *et al.* 2012a] Andrea Calì, Georg Gottlob and Thomas Lukasiewicz. *A general Datalog-based framework for tractable query answering over ontologies*. J. Web Semant., vol. 14, pages 57–83, 2012.

[Calì *et al.* 2012b] Andrea Calì, Georg Gottlob and Andreas Pieris. *Towards more expressive ontology languages: The query answering problem*. Artif. Intell., vol. 193, pages 87–128, 2012.

[Calvanese *et al.* 2005] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini and Riccardo Rosati. *DL-Lite: Tractable Description Logics for Ontologies*. In Manuela M. Veloso and Subbarao Kambhampati, editors, Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA, pages 602–607. AAAI Press / The MIT Press, 2005.

[Calvanese *et al.* 2014] Diego Calvanese, Thomas Eiter and Magdalena Ortiz. *Answering regular path queries in expressive Description Logics via alternating tree-automata*. Inf. Comput., vol. 237, pages 12–55, 2014.

[Ciucanu 2015] Radu Ciucanu. *Cross-Model Queries and Schemas: Complexity and Learning*. PhD thesis, University of Lille, 2015.

[Console & Lenzerini 2014] Marco Console and Maurizio Lenzerini. *Data Quality in Ontology-based Data Access: The Case of Consistency*. In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada, pages 1020–1026, 2014.

[Corman *et al.* 2018] Julien Corman, Juan L. Reutter and Ognjen Savkovic. *Semantics and Validation of Recursive SHACL*. In Denny Vrandecic, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee and Elena Simperl, editors, The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I, volume 11136 of *Lecture Notes in Computer Science*, pages 318–336. Springer, 2018.

[Cruz *et al.* 1987] Isabel F. Cruz, Alberto O. Mendelzon and Peter T. Wood. *A Graphical Query Language Supporting Recursion*. In Umeshwar Dayal and Irving L.

Traiger, editors, Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, CA, USA, May 27-29, 1987, pages 323–330. ACM Press, 1987.

[Das *et al.* 2011] Souripriya Das, Seema Sundara and Richard Cyganiak. *R2RML: RDB to RDF Mapping Language*. `http://www.w3.org/TR/r2rml/`, 2011.

[David *et al.* 2010] Claire David, Leonid Libkin and Filip Murlak. *Certain answers for XML queries*. In Jan Paredaens and Dirk Van Gucht, editors, Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA, pages 191–202. ACM, 2010.

[de Rougemont & Vieilleribière 2007] Michel de Rougemont and Adrien Vieilleribière. *Approximate Data Exchange*. In Database Theory - ICDT 2007, 11th International Conference, Barcelona, Spain, January 10-12, 2007, Proceedings, pages 44–58, 2007.

[Fagin *et al.* 2005a] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller and Lucian Popa. *Data exchange: semantics and query answering*. Theor. Comput. Sci., vol. 336, no. 1, pages 89–124, 2005.

[Fagin *et al.* 2005b] Ronald Fagin, Phokion G. Kolaitis and Lucian Popa. *Data exchange: getting to the core*. ACM Trans. Database Syst., vol. 30, no. 1, pages 174–210, 2005.

[Fagin *et al.* 2009] Ronald Fagin, Laura M. Haas, Mauricio A. Hernández, Renée J. Miller, Lucian Popa and Yannis Velegrakis. *Clio: Schema Mapping Creation and Data Exchange*. In Alexander Borgida, Vinay K. Chaudhri, Paolo Giorgini and Eric S. K. Yu, editors, Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos, volume 5600 of *Lecture Notes in Computer Science*, pages 198–236. Springer, 2009.

[Fuxman *et al.* 2006] Ariel Fuxman, Mauricio A. Hernández, C. T. Howard Ho, Renée J. Miller, Paolo Papotti and Lucian Popa. *Nested Mappings: Schema Mapping Reloaded*. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet,

Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha and Young-Kuk Kim, editors, Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006, pages 67–78. ACM, 2006.

[Garcia-Molina *et al.* 2009] Hector Garcia-Molina, Jeffrey D. Ullman and Jennifer Widom. Database systems - the complete book (2. ed.). Pearson Education, 2009.

[Gayo *et al.* 2017] José Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva and Dimitris Kontokostas. Validating RDF data. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers, 2017.

[Groppe & Groppe 2008] Sven Groppe and Jinghua Groppe. *Output schemas of XSLT stylesheets and their applications*. Inf. Sci., vol. 178, no. 21, pages 3989–4018, 2008.

[Henzinger *et al.* 1995] Monika Rauch Henzinger, Thomas A. Henzinger and Peter W. Kopke. *Computing Simulations on Finite and Infinite Graphs*. In 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995, pages 453–462. IEEE Computer Society, 1995.

[Heyvaert *et al.* 2016] Pieter Heyvaert, Anastasia Dimou, Aron-Levi Herregodts, Ruben Verborgh, Dimitri Schuurman, Erik Mannens and Rik Van de Walle. *RMLEditor: A Graph-Based Mapping Editor for Linked Data Mappings*. In Harald Sack, Eva Blomqvist, Mathieu d'Aquin, Chiara Ghidini, Simone Paolo Ponzetto and Christoph Lange, editors, The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings, volume 9678 of *Lecture Notes in Computer Science*, pages 709–723. Springer, 2016.

[Heyvaert *et al.* 2018] Pieter Heyvaert, Ben De Meester, Anastasia Dimou and Ruben Verborgh. *Declarative Rules for Linked Data Generation at Your Fingertips!* In Aldo Gangemi, Anna Lisa Gentile, Andrea Giovanni Nuzzolese, Sebastian Rudolph, Maria Maleshkova, Heiko Paulheim, Jeff Z. Pan and Mehwish Alam,

editors, The Semantic Web: ESWC 2018 Satellite Events - ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers, volume 11155 of *Lecture Notes in Computer Science*, pages 213–217. Springer, 2018.

[Hutchins *et al.* 1985] Edwin Hutchins, James D. Hollan and Donald A. Norman. *Direct Manipulation Interfaces*. Hum. Comput. Interact., vol. 1, no. 4, pages 311–338, 1985.

[Jung *et al.* 2018] Jean Christoph Jung, Carsten Lutz, Mauricio Martel and Thomas Schneider. *Querying the Unary Negation Fragment with Regular Path Expressions*. In Benny Kimelfeld and Yael Amsterdamer, editors, 21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria, volume 98 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[Junior *et al.* 2017] Ademar Crotti Junior, Christophe Debruyne and Declan O'Sullivan. *Juma: An Editor that Uses a Block Metaphor to Facilitate the Creation and Editing of R2RML Mappings*. In Eva Blomqvist, Katja Hose, Heiko Paulheim, Agnieszka Lawrynowicz, Fabio Ciravegna and Olaf Hartig, editors, The Semantic Web: ESWC 2017 Satellite Events - ESWC 2017 Satellite Events, Portorož, Slovenia, May 28 - June 1, 2017, Revised Selected Papers, volume 10577 of *Lecture Notes in Computer Science*, pages 87–92. Springer, 2017.

[Kellou-Menouer & Kedad 2015] Kenza Kellou-Menouer and Zoubida Kedad. *Schema Discovery in RDF Data Sources*. In Paul Johannesson, Mong-Li Lee, Stephen W. Liddle, Andreas L. Opdahl and Oscar Pastor López, editors, Conceptual Modeling - 34th International Conference, ER 2015, Stockholm, Sweden, October 19-22, 2015, Proceedings, volume 9381 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2015.

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

[Kolaitis 2005] Phokion G. Kolaitis. *Schema mappings, data exchange, and metadata management*. In Chen Li, editor, Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA, pages 61–75. ACM, 2005.

[Lassila & Swick 1999] Ora Lassila and Ralph Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. `https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`, 1999.

[Li *et al.* 2019] Yeting Li, Haiming Chen, Xiaolan Zhang and Lingqi Zhang. *An Effective Algorithm for Learning Single Occurrence Regular Expressions with Interleaving*. In Proceedings of the 23rd International Database Applications & Engineering Symposium, IDEAS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[Lohse *et al.* 1994] Gerald L. Lohse, Kevin Biolsi, Neff Walker and Henry H. Rueter. *A Classification of Visual Representations*. Commun. ACM, vol. 37, no. 12, pages 36–49, 1994.

[Maier *et al.* 1979] David Maier, Alberto O. Mendelzon and Yehoshua Sagiv. *Testing Implications of Data Dependencies*. ACM Trans. Database Syst., vol. 4, no. 4, pages 455–469, 1979.

[Marnette *et al.* 2011] Bruno Marnette, Giansalvatore Mecca, Paolo Papotti, Salvatore Raunich and Donatello Santoro. *++Spicy: an OpenSource Tool for Second-Generation Schema Mapping and Data Exchange*. Proc. VLDB Endow., vol. 4, no. 12, pages 1438–1441, 2011.

[Marnette 2009] Bruno Marnette. *Generalized Schema-Mappings: From Termination to Tractability*. In Proceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '09, pages 13–22, New York, NY, USA, 2009. ACM.

[Milner 1971] Robin Milner. *An Algebraic Definition of Simulation Between Programs*. In D. C. Cooper, editor, Proceedings of the 2nd International Joint Conference

on Artificial Intelligence. London, UK, September 1-3, 1971, pages 481–489. William Kaufmann, 1971.

[Nikolaou & Koubarakis 2016] Charalampos Nikolaou and Manolis Koubarakis. *Querying incomplete information in RDF with SPARQL*. Artif. Intell., vol. 237, pages 138–171, 2016.

[Pérez *et al.* 2010] Jorge Pérez, Marcelo Arenas and Claudio Gutiérrez. *nSPARQL: A navigational language for RDF*. J. Web Semant., vol. 8, no. 4, pages 255–270, 2010.

[Pichler & Savenkov 2009] Reinhard Pichler and Vadim Savenkov. *DEMo: Data Exchange Modeling Tool*. Proc. VLDB Endow., vol. 2, no. 2, pages 1606–1609, 2009.

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

[Raffio *et al.* 2008] Alessandro Raffio, Daniele Braga, Stefano Ceri, Paolo Papotti and Mauricio A. Hernández. *Clip: a Visual Language for Explicit Schema Mappings*. In Gustavo Alonso, José A. Blakeley and Arbee L. P. Chen, editors, Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico, pages 30–39. IEEE Computer Society, 2008.

[Rodriguez-Muro & Rezk 2015] Mariano Rodriguez-Muro and Martín Rezk. *Efficient SPARQL-to-SQL with R2RML mappings*. J. Web Semant., vol. 33, pages 141–169, 2015.

[Sengupta *et al.* 2013] Kunal Sengupta, Peter Haase, Michael Schmidt and Pascal Hitzler. *Editing R2RML Mappings Made Easy*. In Eva Blomqvist and Tudor Groza, editors, Proceedings of the ISWC 2013 Posters & Demonstrations Track, Sydney, Australia, October 23, 2013, volume 1035 of *CEUR Workshop Proceedings*, pages 101–104. CEUR-WS.org, 2013.

[Sequeda *et al.* 2012] Juan F. Sequeda, Marcelo Arenas and Daniel P. Miranker. *On directly mapping relational databases to RDF and OWL*. In Alain Mille, Fabien L. Gandon, Jacques Misselis, Michael Rabinovich and Steffen Staab, editors, Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012, pages 649–658. ACM, 2012.

[Shneiderman 1983] Ben Shneiderman. *Direct Manipulation: A Step Beyond Programming Languages*. Computer, vol. 16, no. 8, pages 57–69, 1983.

[Sicilia *et al.* 2017] Álvaro Sicilia, German Nemirovski and Andreas Nolle. *Map-On: A web-based editor for visual ontology mapping*. Semantic Web, vol. 8, no. 6, pages 969–980, 2017.

[Staworko & Wieczorek 2019] Slawek Staworko and Piotr Wieczorek. *Containment of Shape Expression Schemas for RDF*. In Dan Suciu, Sebastian Skritek and Christoph Koch, editors, Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019, pages 303–319. ACM, 2019.

[Staworko *et al.* 2015] Slawek Staworko, Iovka Boneva, José Emilio Labra Gayo, Samuel Hym, Eric G. Prud'hommeaux and Harold R. Solbrig. *Complexity and Expressiveness of ShEx for RDF*. In Marcelo Arenas and Martín Ugarte, editors, 18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium, volume 31 of *LIPIcs*, pages 195–211. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[Thompson *et al.* 2012] Henry Thompson, Noah Mendelson, David Beech and Murray Maloney. *XML Schema Definition Language (XSD)*. `https://www.w3.org/TR/xmlschema11-1/`, 2012.

[Treisman 1985] Anne Treisman. *Preattentive processing in vision*. Comput. Vis. Graph. Image Process., vol. 31, no. 2, pages 156–177, 1985.

[Tzitzikas *et al.* 2012] Yannis Tzitzikas, Christina Lantzaki and Dimitris Zeginis. *Blank Node Matching and RDF/S Comparison Functions*. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Man-

fred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein and Eva Blomqvist, editors, The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I, volume 7649 of *Lecture Notes in Computer Science*, pages 591–607. Springer, 2012.

[Vardi 1998] Moshe Y. Vardi. *Reasoning about The Past with Two-Way Automata*. In Kim Guldstrand Larsen, Sven Skyum and Glynn Winskel, editors, Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.

[Vavliakis *et al.* 2013] Konstantinos N. Vavliakis, Theofanis K. Grollios and Pericles A. Mitkas. *RDOTE - Publishing Relational Databases into the Semantic Web*. J. Syst. Softw., vol. 86, no. 1, pages 89–99, 2013.

[Villazón & Hausenblas 2012] Boris Villazón and Michael Hausenblas. *R2RML and Direct Mapping Test Cases*, 2012.

[W3C 2013] W3C. *RDF Validation Workshop Report: Practical Assurances for Quality RDF Data*. `http://www.w3.org/2012/12/rdf-val/report`, 2013.

[Winskel 1993] Glynn Winskel. The formal semantics of programming languages - an introduction. Foundation of computing series. MIT Press, 1993.