Ontology-Mediated Queries for NOSQL Databases

Marie-Laure Mugnier
Université de Montpellier
CNRS, LIRMM
GrahpIK Team, INRIA
marie-laure.mugnier@lirmm.fr

Marie-Christine Rousset
Univ. Grenoble Alpes
CNRS, LIG
IUF (Institut Universitaire de France)
marie-christine.rousset@imag.fr

Federico Ulliana
Université de Montpellier
CNRS, LIRMM
GrahpIK Team, INRIA
federico.ulliana@lirmm.fr

Abstract

Ontology-Based Data Access has been studied so far for relational structures and deployed on top of relational databases. This paradigm enables a uniform access to heterogeneous data sources, also coping with incomplete information. Whether OBDA is suitable also for non-relational structures, like those shared by increasingly popular NOSQL languages, is still an open question. In this paper, we study the problem of answering ontology-mediated queries on top of *key-value stores*. We formalize the data model and core queries of these systems, and introduce a rule language to express lightweight ontologies on top of data. We study the decidability and data complexity of query answering in this setting.

Introduction

Ontology-based data access (OBDA) is a well-established paradigm for querying incomplete data sources while taking into account knowledge provided by a domain ontology (Poggi et al. 2008). Today, the main applications of OBDA can be found in data integration as well as in querying the Semantic Web. The interest of OBDA is to allow the users to ask queries on high-level ontology vocabularies and to delegate to algorithms (1) the reformulation of these high-level queries into a set of low-level databases queries, (2) the efficient computation of their answers by native data management systems in which data is stored and indexed, and (3) the combination of these answers in order to obtain the final answers to the users' query. The advantage of OBDA is that, since the query reformulation step is independent of the data, ontology-mediated query answering has the same *data complexity* as the query engines equipping the underlying native data storage systems, and can benefit from the many low-level optimizations making them efficient and scalable. OBDA has been studied so far for relational structures and deployed on top of relational databases. The database queries used in OBDA are (unions of) conjunctive *relational* queries, while the ontologies are specified in either a description logic (e.g., the lightweight DL-Lite (Calvanese et al. 2007)(Kontchakov et al. 2010), or the expressive Horn-$\mathcal{SHIQ}$ (Eiter et al. 2012)), or, more generally, a suitable fragment of first-order logic (e.g., Datalog$\pm$ (Calì, Gottlob, and Pieris 2012) and existential rules (Baget et al. 2011)). Within this framework, decidability and complexity results have been obtained for ontology-mediated query answering, and many algorithms have been designed and implemented.

Whether this paradigm can be used to query datasources that are not relational is a still an open question. The naive way to deal with non relational datasources is to define mappings for translating them into relational structures, and then use the classic OBDA framework as it is. However, this approach would induce a significant performance degrade as it would add a step for converting the data using the mappings and, most importantly, it would make impossible to take advantage of the low level query optimizations provided by native systems. This can be particularly acute for NOSQL systems, like *key-value stores*, that have been specifically designed to scale when dealing with very large collections of data.

In this paper, we study ontology-based data access directly on top of NOSQL systems. The term NOSQL (NotOnly SQL) defines a broad collection of languages. Key-value stores are NOSQL systems adopting the data model of *key-value records* (also called JSON records). These records are processed on distributed systems, but also increasingly exchanged on the Web thereby replacing semistructured XML data and many RDF formats (see JSON-LD (Sporny et al. 2004)). Key-value records are non-first normal forms where values are not only atomic (in contrast with relational databases) and nesting is possible (Abiteboul, Hull, and Vianu 1995).

{ department : *"Computer Science"*,
  professor : [
  { name : *"Alice"*, reachable : *"yes"*,
    boss : *"Charles"* }
  { name : *"Bob"*, phone : {office : *"5-256"*} }]
  course : [[*"C123"*, *"Java"*] , [*"C310"*, *"C++"*]]
  director : *null* }

Figure 1: Key-value record

Illustrative example

To illustrate, consider the record in Figure 1. It contains nine different keys among which five (department, name, reachable, boss, office) are associated with basic values. The key professor is associated with a sequence of two basic records, the key course with a sequence of two (nested) sequences, and the key director with an unspecified value $null$. Key-value (KV) stores feature a limited set of low-level operations to query and update keys and values, namely

$\mathsf{get}(k)$ $\mathsf{put}(k,v)$ $\mathsf{clear}(k)$

These systems are designed and optimized for this kind of operations that can be performed in parallel over large collections of records, and leave other functionalities (such as joins between records) for the application accessing the NOSQL database.

In this work we are interested in *querying* KV stores with ontologies and we mainly focus on get() queries. An example of query retrieving names of professors in a key-value record is $\mathsf{get}(\mathit{professor.name})$. This expression returns the values "*Alice*" and "*Bob*" once evaluated on the record of Figure 1. Another feature supported by KV stores is the possibility to check structural properties of a record before selecting some content. Thus, we further look at queries with a check() construct like $\mathsf{check}(\mathit{director}).\mathsf{get}(\mathit{professor.name})$. The get part of the query is evaluated at the only condition that the key director also exists in the record. The check() construct is particularly useful when dealing with incomplete information (values for which we just know the existence) which is expressed by null values, like for the key director in Figure 1.

Our goal is to study how accessing information on KV stores can be enhanced by ontology reasoning. To illustrate the role of ontologies, consider the query $\mathsf{get}(\mathit{professor.contact.office})$, searching through the contacts of all professors. Although in the record of Figure 1 there is a phone record this query yields no result because key contact is not found. However, by introducing some general knowledge saying that a phone number is a contact we could output a result for the query. Here is where ontology-based data access comes into play. In fact, we could simply equip the store with a rule saying that "*any value for the key* phone *is also a value for the key* contact":

$$\texttt{phone}(x) \longrightarrow \texttt{contact}(x)$$

thereby retrieving the value associated to the key office in the subrecord. Furthermore, rules able to assert the existence of a value can be used to reason on incomplete information. For example, the following rule says that whenever we find a value for the key director in a record, there exists a value for the key assistant (although this could be unspecified):

$$\texttt{director}(x) \longrightarrow \exists y.\, \texttt{assistant}(y)$$

With this knowledge in hand, the evaluation of the query check(assistant).get(department) on the record of Figure 1 outputs "*Computer Science*", although there is no explicit value for the key assistant.

From a practical point of view, the interest of implementing OBDA over KV stores is that, like the queries, inference rules should be *parallelizable* over all records.

Contributions

This paper makes the following contributions towards the study of OBDA for NOSQL databases. First, we provide a formal syntax and semantics for data and core queries based on homomorphisms, supported by popular key-value stores like MongoDB - which is missing (Ong, Papakonstantinou, and Vernoux 2014). Second, we present a rule language with clear formal semantics suitable for writing lightweight ontologies on top of KV records. Finally, we give a sound and complete reformulation technique, as well as (un)decidability and data complexity results for ontology-mediated query answering in this setting.

Formal Background

In this section, we formalize the data model of key-value stores, and the associated query language.

Key-value Records Let CONST be an infinite set of data constants and NULLS be an infinite set of *nulls* used as placeholders for unknown values. A *key-value record* r is a finite and non-empty set of key-value pairs of the form (k,v), each assigning the value v to key k. Furthermore, any key k occurs at most once in this set. Values are terms generated by the following grammar

$$r ::= \{\ (k,v)\ \ldots\ (k,v)\ \} \quad v ::= a \mid \mathit{null} \mid [v \ldots v] \mid r$$

with $a \in$ CONST, $\mathit{null} \in$ NULLS, and where $[v \ldots v]$ is a non-empty sequence. A record r is an *unordered* set of key-value pairs. RECS denotes the set of records that can be recursively built in this way. The set of keys at the *top level* of a record $r = \{(k_1,v_1), \ldots, (k_n,v_n)\}$,

denoted by $keys(r)$, is defined as $keys(r)=\{k_1,...,k_n\}$. Then, $value(r,k_i) = v_i$, if $k_i \in keys(r)$, and is undefined otherwise. We say that a value v is *terminal* when $v \in$ CONST, or v is a nested sequence, i.e., a sequence within a sequence. Indeed, nested sequences are never navigated inside by the queries considered in this paper. A *key-value store* I is a finite set of key-value records.

Paths and homomorphisms are the basis of query and rule semantics. A path π in a record r is a word alternating values and keys of the form $\pi=v_0.k_1 \dots k_n.v_n$ such that (a) v_i is a subrecord of r, for all $i < n$ (b) $k_{i+1} \in keys(v_i)$ and (c) if $value(v_i,k_{i+1})$ is a sequence, then v_{i+1} is an element of $value(v_i,k_{i+1})$, otherwise $v_{i+1} = value(v_i,k_{i+1})$ $(0 \leq i < n)$. When $v_0 = r$ the path is said to be *rooted*.

A *key-path* κ is a sequence of keys $\kappa = k_1.k_2 \dots k_n$. A *path-atom* $\kappa(x)$ is such that κ is a key-path and x a variable. We define its *expansion* as $\bar{\kappa}(x) = x_0.k_1.x_1.k_2.x_2 \dots x_{n-1}.k_n(x)$ and $vars(\bar{\kappa}(x))$ as the set $\{x_1,\dots,x_{n-1},x\}$ of its pairwise distinct variables.

A *homomorphism* from $\kappa(x)$ to a record r is a substitution h of the variables in $vars(\bar{\kappa}(x))$ such that $h(\overline{\kappa}(x))=h(x_0).k_1.h(x_1).k_2.h(x_2)\dots h(x_{n-1}).k_n.h(x)$ is a path π in r. If $h(x_0) = r$, then the homomorphism h is said to be *rooted*.

A *ground* path-atom $\kappa(v)$ is such that κ is a key-path and v a value. Its associated record, denoted by $record(\kappa(v))$, is defined as follows. If $\kappa(v) = k(v)$, then $record(\kappa(v)) = \{k : v\}$, otherwise $\kappa(v)$ is of the form $k.\kappa'(v)$ and $record(\kappa(v)) = \{k : record(\kappa'(v))\}$.

Record equality In the formal development, we consider a set-based equality for records, thereby ignoring order inside sequences. Two atomic values are identical, denoted by $v_1 \cong v_2$, if they are the same terminal value or both $null$. The sequence $v = [v_1 \dots v_n]$ is contained in the sequence $v' = [v'_1 \dots v'_m]$, if for each value v_i there exists a v'_j such that $v_i \cong v'_j$. This is denoted by $v \subseteq v'$. We write $v \cong v'$ when $v \subseteq v'$ and $v' \subseteq v$ both hold. Finally, two records are identical, $r_1 \cong r_2$, when $keys(r_1) = keys(r_2)$ and $value(r_1,k) \cong value(r_2,k)$ for each $k \in keys(r_1)$.

Record merging We recursively define an associative operator "$\circ$" for merging two values v and v', which will be useful to define rule semantics.

• In the basic case where the merge involves null values, we define $v \circ null = null \circ v = v$.

We now turn to the cases where $v, v' \neq null$.

• *If v is a constant*, we have to consider three subcases.
- When $v' \cong v$, we define $v \circ v' = v$.
- When $v' \in$ CONST $\cup$ RECS, we define $v \circ v' = [v, v']$.
- When $v' = [v'_1, \dots, v'_n]$, we define $v \circ v' = v'$ if $v \cong v'_i$ for some $v'_i \in v'$, and $v \circ v' = [v, v'_1, \dots, v'_n]$ otherwise. For instance, $1 \circ [\{\texttt{name} : \textit{Alice}\}]=[1, \{\texttt{name} : \textit{Alice}\}]$. The dual cases where v' is a constant but v is not are obtained by exchanging the roles of v and v' respectively.

• *When v and v' are both records* the values of their common keys (denoted by the set C) are merged:

$$v \circ v' = \bigcup_{k_0 \in C} \{(k_0, value(v,k_0) \circ value(v',k_0))\} \cup \bigcup_{k \in keys(v) \setminus C} \{(k, value(v,k))\} \cup \bigcup_{k' \in keys(v') \setminus C} \{(k', value(v',k'))\}$$

For instance, $\{\texttt{name} : \textit{Bob}, \texttt{age} : \textit{33}\} \circ \{\texttt{name} : \textit{Smith}, \texttt{city} : \textit{Paris}\}=\{\texttt{name} : [\textit{Bob}, \textit{Smith}], \texttt{age} : \textit{33}, \texttt{city} : \textit{Paris}\}$.

• *When v is the sequence $[v_1, \dots, v_n]$ and v' is a record* we have again two subcases to consider.
- When none of the elements in v is a record we define $[v_1, \dots, v_n] \circ v' = [v_1, \dots, v_n, v']$.
- Otherwise, each record in the sequence v is pairwise merged with v'. Hence, $[v_1, \dots, v_n] \circ v' = [v'_1, \dots, v'_n]$ with $v'_i = v_i$ if $v_i \notin$ RECS and $v'_i = (v_i \circ v')$ if $v_i \in$ RECS.

For instance, $[\{\texttt{name} : \textit{Smith}\}, \{\texttt{name} : \textit{Jones}\}] \circ \{\texttt{name} : \{\texttt{first} : \textit{Bob}\}\}$ yields as result the sequence $[\ \{\ \texttt{name} : [\textit{Smith}, \{\texttt{first} : \textit{Bob}\}]\ \}, \{\texttt{name} : [\textit{Jones}, \{\texttt{first} : \textit{Bob}\}]\}]$. Again, the dual case where v is a record and v' is a sequence is obtained by exchanging their roles.

• *Finally, if v and $v' = [v'_1 \dots v'_m]$ are both sequences*, then $v \circ [v'_1 \dots v'_m] = (v \circ v'_1) \circ [v'_2 \dots v'_m]$. For instance, $[1,2] \circ [1,[1]] = [1,2] \circ [[1]] = [1,2,[1]]$.

The merge operator enjoys the following property.

Proposition 1 *$\circ$-merge is commutative wrt $\cong$-equality, i.e., for any pair of values v, v' it holds that $v \circ v' \cong v' \circ v$.*

Queries The query language we consider features standard NOSQL selection and projection on the record structure that we formalize by means of key-paths.

A *get-query* is of the form $Q = \mathsf{get}(\kappa)$ where κ is a key-path specifying a projection expression to retrieve the *terminal values* of interest within records. Its set of answers over a record r is defined as

$$Q(r) = \{h(x) | h \text{ is a rooted-homomorphism from } \kappa(x) \text{ to } r \text{ such that } h(x) \text{ is } \textit{terminal}\}$$

A *check-get query* Q adds a selection condition of the form $Q' = \mathsf{check}(\kappa).\mathsf{get}(\kappa')$ with κ and κ' key-paths. The set of answers of Q on r is defined as follows. Let $Q' = \mathsf{get}(\kappa')$ then $Q(r) = Q'(r)$ if there exists a rooted homomorphism from $\kappa(y)$ to r, otherwise $Q(r) = \emptyset$. Finally, the answer set of a (check-)get query Q over a key-value store I is defined as $Q(I) = \bigcup_{r \in I} Q(r)$.

The NO-RL Rule Language

The language for reasoning on key-value stores, we call NO-RL, is made of *linear* rules where a single path-atom in the body and in the head of rules is allowed. We consider two kinds of NO-RL rules, namely $\forall$-*rules* of the form $\kappa'(x) \longrightarrow \kappa(x)$ and $\exists$-*rules* of the form $\kappa'(x) \longrightarrow \exists y.\kappa(y)$, whose semantics is presented next. In the formal development, we denote a rule by σ a set of rules by Σ. We define three NO-RL profiles of different expressivity.

NO-RL(1) Rules : Key-Atoms

This basic reasoning language consists of rules of the form: $k'(x) \longrightarrow k(x) \mid \exists y.k(y)$ where k, k' are keys (i.e., key-paths of length one). This language allows one to express rules at the record level as the following

$\sigma_1 : \texttt{phone}(x) \longrightarrow \texttt{contact}(x)$

$\sigma_2 : \texttt{boss}(x) \longrightarrow \texttt{reference}(x)$

$\sigma_3 : \texttt{reachable}(x) \longrightarrow \exists y.\texttt{phone}(y)$

The application of the rules on the record of Figure 1 yields the record below.

{ department : *"Computer Science"*,
  professor : [
   {name : *"Alice"*, reachable : *"yes"*,
    boss : *"Charles"*, reference : *"Charles"*,
    phone : *null*, contact : *null*}
   {name : *"Bob"*,
     phone : {office : *"5-256"*}
    contact : {office : *"5-256"*}}]
  course : [[*"C123"*, *"Java"*], [*"C310"*, *"C++"*]]
  director : *null* }

As a main consequence of rule application, the data to take into account when querying the record are not only the input records but also all the data that can be derived from them and the rules.

NO-RL(2): Body Path-Atoms

We now extend the former language by allowing path-atoms in the rule body (only), so as to obtain rules of the form $\kappa(x) \longrightarrow k(x) \mid \exists y.\, k(y)$, like for instance

$\sigma_1 : \texttt{professor.name}(x) \longrightarrow \exists y.\, \texttt{secretary}(y)$

$\sigma_2 : \texttt{professor.boss}(x) \longrightarrow \texttt{director}(x)$

Differently from the previous case, the application of these rules requires some navigation to determine the subrecords to extend with new keys or new values. The following record corresponds to the application of rules, again on the record of Figure 1.

{ department : *"Computer Science"*,
  professor : [
  {name : *"Alice"*, reachable : *"yes"*,
   boss : *"Charles"* },
  {name : *"Bob"*, phone : {office : *"5-256"*}}]
  course : [[*"C123"*, *"Java"*], [*"C310"*, *"C++"*]]
  secretary : *null*, director : *"Charles"* }

NO-RL(3): Head Path-Atoms

This third language is the dual of NO-RL(2) as path-atoms are allowed in the head of rules (only), thereby yielding rules of the form $k(x) \longrightarrow \kappa(x) \mid \exists y.\, \kappa(y)$ like

$\texttt{department}(x) \longrightarrow \texttt{professor.specialty}(x)$

saying that the department determines the teaching specialty of all of its professors. Applying this NO-RL(3) rule on the record of Figure 1 gives the following result.

{ department : *"Computer Science"*,
  professor : [
   {name : *"Alice"*, reachable : *"yes"*, boss : *"Charles"*, specialty : *"Computer Science"*},
   {name : *"Bob"*, phone : {office : *"5-256"*},
    specialty : *"Computer Science"*}]
  course : [[*"C123"*, *"Java"*], [*"C310"*, *"C++"*]]
  director : *null* }

Definition 1 (NO-RL) NO-RL *is obtained as union of* NO-RL(3)*,* NO-RL(2) *(and thus* NO-RL(1)*) languages.*

More general rules of the form $k'_1.k'_2 \ldots k'_n(x) \longrightarrow k_1.k_2 \ldots k_m(y)$, with $n, m \geq 2$ (and possibly $x = y$), can be simulated by a pair of two NO-RL rules, namely $k'_1.k'_2 \ldots k'_n(x) \rightarrow k_0(x)$ and $k_0(x) \rightarrow k_1.k_2 \ldots k_m(y)$.

NO-RL Rule Semantics

The formal semantics of NO-RL rules defines the effect of rule application on a KV store. This relies on the merge operator (see Section "Formal Background"), which is used to enrich a record with either values copied from its subrecords, or with fresh nulls. We first define the single-step application of a NO-RL rule on a record, and then the usual inference operator.

Definition 2 (Rule Semantics) *Let σ : Body$(x) \longrightarrow$ Head$(x) \mid \exists y.$ Head(y) be a NO-RL rule and r a record. The rule σ is said to be* applicable *on r if there is a homomorphism h from Body(x) to r. We denote by r_i the subrecord of r at the root of the path $h($Body$(x))$.*

The application of σ (w.r.t. h) to r consists of merging the subrecord r_i with a fresh record r_σ defined as

$$r_\sigma = \begin{cases} \textit{record}(\textit{Head}(h(x))) & \textit{if } \sigma \textit{ is a } \forall\textit{-rule} \\ \textit{record}(\textit{Head}(\textit{null})) & \textit{if } \sigma \textit{ is an } \exists\textit{-rule} \end{cases}$$

Let r' be the obtained record. Then, the single step application of σ on r (wrt h) is denoted by $\sigma, r \vdash_1 r'$.

The inference operator (denoted by $\vdash$) consists of successive rule application steps. Given a set of rules Σ and a record r, we write $r, \Sigma \vdash r'$ if there are $r_1, \ldots, r_n$ and $\sigma_1, \ldots, \sigma_n \in \Sigma$ s.t. $\sigma_i, r_i \vdash_{\mathbf{1}} r_{i+1}$ with $r \cong r_1$ and $r' \cong r_n$ $(1 \leq i < n)$. Finally, we write $I, \Sigma \vdash r'$ when there exists $r \in I$ such that $r, \Sigma \vdash r'$.

Query semantics under NO-RL rules is now defined. A value is an answer to a query Q over a set of key-values records and a set of rules if it can be obtained as a result of the evaluation of the query Q over one record inferred from one input record by application of the rules.

Definition 3 (Query Semantics Under NO-RL Rules) *Given Q, I and Σ we define $Q(I, \Sigma) = \bigcup_{I, \Sigma \vdash r'} Q(r')$.*

A desirable property of any inference system is that the order in which rules are applied is irrelevant. NO-RL rules enjoy this property, which relies on the commutativity of the merge operator up to $\cong$-equality (Proposition 1) and to the monotonicity of rule application. This last one ensures that if r' is obtained from r in one-step then every rule application that was possible on r is still possible on r'.

Theorem 1 (Confluence) *For any record r, if $r, \Sigma \vdash r_1$ and $r, \Sigma \vdash r_2$ with $r_1 \ncong r_2$ then there are $\bar{r}_1$ and $\bar{r}_2$ such that $r_1, \Sigma \vdash \bar{r}_1$ and $r_2, \Sigma \vdash \bar{r}_2$ with $\bar{r}_1 \cong \bar{r}_2$.*

To illustrate, consider the record r=*record*$(\mathtt{a.b}(v))$ and the rules $\sigma_1 : \mathtt{a}(x) \longrightarrow \mathtt{c}(x)$ and $\sigma_2 : \mathtt{b}(x) \longrightarrow \mathtt{d}(x)$. Applying σ_1 first gives $r_1 = \{\mathtt{a} : \{\mathtt{b} : v\}, \mathtt{c} : \{\mathtt{b} : v\}\}$. Applying σ_2 first gives $r_2 = \{\mathtt{a} : \{\mathtt{b} : v, \mathtt{d} : v\}\}$. Of course these intermediate records are different, $r_1 \ncong r_2$. However, the fixpoint application of the rules is confluent on record $\bar{r}$=$\{\mathtt{a} : \{\mathtt{b} : v, \mathtt{d} : v\}, \mathtt{c} : \{\mathtt{b} : v, \mathtt{d} : v\}\}$. Indeed, $\bar{r}$ can be obtained (i) from r_2 by simply applying σ_1 or (ii) from r_1 by applying σ_2 first, and then σ_1. or (iii) from r_1 by applying σ_2 two consecutive times.

We refer to *saturation* as the process of applying a set of NO-RL rules Σ on a record r, until fixpoint. Notice that, as a corollary of Theorem 1, when this process is finite, it yields a unique record modulo $\cong$-equality, we note r_Σ. In this case, $Q(r, \Sigma) = Q(r_\Sigma)$ for any $r \in I$.

Interestingly enough, because queries extract only terminal values from records, $\exists$-rules do not intervene in the computation of answers to get-queries.

Proposition 2 *For any* get-query $Q(I, \Sigma) = Q(I, \Sigma_\forall)$ *where $\Sigma_\forall$ is the restriction of Σ to $\forall$-rules.*

This is however not the case for general (check-get) queries which need both kinds of rules.

Negative Results

We now study the decidability and complexity of the following fundamental Query Answering (QA) decision problem.

Definition 4 (Query Answering Decision Problem) *Given I, Σ, Q, and a value v, does $v \in Q(I, \Sigma)$?*

Despite the fact that NO-RL rules seem simple, we prove an undecidability result for general NO-RL rules, which already holds for get queries.

Theorem 2 *QA is undecidable for NO-RL rules.*

Proof:(sketch) By reduction from the word problem in a semi-Thue system, which is known to be undecidable. Given two words w, w_f and a set of rewriting rules Γ, the word problem asks if there is a derivation of w_f from w with the rules in Γ. The reduction establishes a bijective relation between the set of symbols in w, w_f, and Γ, and the set of keys employed in a record. We build an instance $I = \{r\}$, where $r =$ *record*$(\kappa_w(v))$, with κ_w the key-path corresponding to w and v a special constant. Then, each rewriting rule $w_2 \leftarrow w_1 \in \Gamma$ becomes a NO-RL inference rule $w_1(x) \longrightarrow w_2(x) \in \Sigma$. Note that Σ may contain both NO-RL(2) and NO-RL(3) rules. Finally, we define $Q = \mathsf{get}(\kappa_{w_f})$, with κ_{w_f} the key-path associated with w_f. We can then establish a bijective correspondence between (i) any derivation from w with Γ and (ii) the set of rooted-paths ending with the terminal value v of a record inferred from r and Σ. It follows that w_f is derivable from w with Γ iff $v \in Q(I, \Sigma)$. $\square$

We can also see that as soon a *single* NO-RL(3) rule is involved, saturation may be infinite, as illustrated by the following example. On the record $\{\mathtt{a} : v\}$, the rule $\sigma = \mathtt{a}(x) \longrightarrow \mathtt{b.a}(x)$ can be applied indefinitely, thereby yielding an unbounded number of records:

$\{\mathtt{a} : v, \mathtt{b} : \{\mathtt{a} : v\}\}$
$\{\mathtt{a} : v, \mathtt{b} : \{\mathtt{a} : v, \mathtt{b} : \{\mathtt{a} : v\}\}\}$
$\{\mathtt{a} : v, \mathtt{b} : \{\mathtt{a} : v, \mathtt{b} : \{\mathtt{a} : v, \mathtt{b} : \{\mathtt{a} : v\}\}\}\}$ ...

If we consider only NO-RL(2) (and thus NO-RL(1)) rules, the saturation is always finite because NO-RL(2) rules do not increase the depth of records. However, already for NO-RL(1) rules, the size of the saturation can be exponential in the size of the data. Consider for instance the rule $\sigma : k(x) \longrightarrow k_1(x)$ and the record $r =$ *record*$(k.k \ldots k(v))$ of depth d. The exhaustive application of σ gives us a record representing a complete binary tree of depth d.

These observations suggest that saturating the store may not be the most suitable approach to query answering. This leads us to turn towards query reformulation techniques, in the spirit of the OBDA paradigm, where data access is fully delegated to the database system.

Query Answering based on Reformulation

Query reformulation amounts to finding a set of queries $\{Q_1, \ldots, Q_n\}$ which is equivalent to a given query Q wrt a set of rules Σ over all possible KV store instances.

As already discussed in the introduction, the main advantage of the approach is that it accesses information leaving untouched the original data, avoiding extra storage and maintenance costs and reusing database technology.

To see how reformulation works consider the query Q : get($\texttt{professor.contact.office}$) and the rule σ_1 : $\texttt{phone}(x) \longrightarrow \texttt{contact}(x)$. A reformulation of Q with σ_1 is Q' : get($\texttt{professor.phone.office}$). This expression can be evaluated directly on the original data (for instance on the record of Figure 1) together with Q, without requiring any rule application. Consider now an $\exists$-rule like σ_2 : $\texttt{reachable}(x) \longrightarrow \exists y.\texttt{phone}(y)$. This rule cannot be used for reformulating the query get($\texttt{professor.phone}$), for which the expression get($\texttt{professor.reachable}$) is of course not a reformulation. Nevertheless, σ_2 is helpful for reformulating Boolean parts of queries. For example, a Boolean condition like check($\texttt{professor.reachable}$) will be a reformulation of check($\texttt{professor.phone}$).

To make reformulation effective we need two kinds of path-reformulations.

- Given the path-atom $\kappa(x) = \kappa_1.\kappa_2.\kappa_3(x)$ (with κ_1, κ_3 possibly empty) and the rule σ : $\kappa_0(x_0) \longrightarrow \kappa_2(x_0)$ we say that κ' is a *value path-reformulation* of κ if $\kappa' = \kappa_1.\kappa_0.\kappa_3(x)$
- Given the path-atom $\kappa(x) = \kappa_1.\kappa_2(x)$ and σ : $\kappa_0(x_0) \longrightarrow \kappa_2.\kappa_3(y)$ (where possibly $x_0 = y$ and κ_1, κ_3 are empty), we say that κ' is a *Boolean path-reformulation* of κ with σ if $\kappa' = \kappa_1.\kappa_0(x_0)$.

Definition 5 (Query Reformulation) *We say that* $Q' = \textit{check}(\kappa'_2).\textit{get}(\kappa'_1)$ *is a* direct *reformulation of* $Q = \textit{check}(\kappa_2).\textit{get}(\kappa_1)$ *with* σ *if either* κ'_1 *is a value path-reformulation of* κ_1 *with* σ *or* κ'_2 *is a (value or Boolean) path-reformulation of* κ_2 *with* σ.

Furthermore, Q' *is a reformulation of* Q *wrt* Σ *if there exists a finite sequence of queries* $(Q_1, \ldots, Q_n)$ *such that* $Q_1 = Q$, $Q_n = Q'$ *and for each* $1 \leq i < n$ Q_{i+1} *is a reformulation of* Q_i *wrt* Σ. *The set of reformulations of* Q *wrt* Σ *is denoted by* $\textsc{Ref}(Q, \Sigma)$. *Note that* $Q \in \textsc{Ref}(Q, \Sigma)$.

We first prove the soundness and completeness of QA based on query reformulation.

Theorem 3 (Soundness and Completeness) $Q(I, \Sigma) = \bigcup_{Q' \in \textsc{Ref}(Q,\Sigma)} Q'(I)$

Proof: (sketch) This follows from the correspondence between one rule application and one reformulation step of *either* the check or the get path-atom of the query. The correspondence takes into account value and Boolean reformulations in slightly different ways. □

Data Complexity of QA

Having a sound and complete reformulation procedure gives us directly a query answering algorithm for the case where $\textsc{Ref}(Q, \Sigma)$ is finite. For NO-RL(3), and thus NO-RL(1), it is indeed the case. For NO-RL(1), this follows from the fact that a rule always replaces a key with at most one key thereby keeping constant the size of the paths in the reformulations. As the symbols used in the query are also finite, we have the finiteness of the reformulation. More precisely, the number of reformulations of a query is in $O(|Q|^{|\Sigma|})$ as each symbol in a path of Q can be chosen by at most $|\Sigma|$ keys. For strictly NO-RL(3) rules, notice that reformulation always lowers the size of the paths in the query. Since the set $\textsc{Ref}(Q, \Sigma)$ is always finite for NO-RL(3) rules, the data-complexity of QA under NO-RL(3) rules is the same as basic query answering on KV stores, which we prove to be in the low complexity class AC^0.

Theorem 4 *Query answering on KV-stores (without rules) is in* AC^0 *for data complexity.*

Proof: (sketch) By reduction to the problem of answering conjunctive queries (actually "paths") on relational databases, known to have AC^0 data complexity. □

Corollary 1 *QA under NO-RL(3) (and thus NO-RL(1)) rules is in* AC^0 *for data complexity.*

Proof: This follows from the fact that reformulation is in constant time for data complexity and Theorem 4. □

We now turn our attention to NO-RL(2). In a dual way to saturation under NO-RL(3) rules, $\textsc{Ref}(Q, \Sigma)$ can be infinite under NO-RL(2) rules. Indeed, consider the rule σ : $\texttt{a.b}(x) \longrightarrow \texttt{a}(x)$ and the query $Q =$ get($\texttt{a}$) which has an infinite reformulation set.

get($\texttt{a.b}$) get($\texttt{a.b.b}$) get($\texttt{a.b.b.b}$) ...

However, it turns out that not all of these reformulations are useful for QA when the instance I is fixed. Given any record r of depth d, every *useful* reformulation of Q, i.e., potentially able to find an answer in r, has length bounded by d. Since a NO-RL(2)-reformulation of a path κ cannot be of smaller length than $|\kappa|$, completeness is not harmed if we exclude all reformulations with length greater than d. There is a finite (although exponential in d) number of useful reformulations. Beside, the number of steps needed to produce a reformulation is bounded by $(d + 1) \times |\Sigma|$. Indeed, strict NO-RL(2) rules can be used at most d times to obtain a path of length bounded by d, interleaved by at most one application of each NO-RL(1) rule. This observation lead us to the following result.

Theorem 5 *QA under NO-RL(2) rules is in NP for data complexity.*

Summing up, the QA problem on KV stores with NO-RL(3) (and thus NO-RL(1)) rules belongs to a low

data complexity class. Since this result is based on reformulation, this technique is likely to be efficiently implementable. For NO-RL(2) rules reformulation can still be computed in non-deterministic polynomial time.

Related Work and Conclusion

We presented a rule-based framework for answering ontology-mediated queries over KV stores. We shown that despite the apparent simplicity of our language, query answering surprisingly turns out undecidable. We identified several fragments of rules for which the query answering problem has a low (data) complexity, motivating the implementation of our framework on top of NOSQL databases. Up to our knowledge, this work is the first one that lays the formal basis of ontolgy-based data access on top of NOSQL databases and that provides algorithms of query reformulation in this setting.

The NO-RL rule language operates on key-value records and can be compared to existing languages designed for reasoning on nested structures. For instance, Frame Logic (Kifer, Lausen, and Wu 1995; Kifer 2005) provides a logical foundation for frame-based and object-oriented languages for data and knowledge representation. Its expressivity captures the NO-RL language but there are no computational guarantees. The Elog rule language (Baumgartner et al. 2001) underlying the the Lixto system (Baumgartner et al. 2003) is a fragment of monadic Datalog (Gottlob and Koch 2004) that has been specifically designed for extracting tree shaped data from HTML pages. These rules are similar to $\forall$-rules of the NO-RL(2) fragment. They are used in forward-chaining manner to generate novel logical structures that are then exported in XML. Active XML (Abiteboul, Benjelloun, and Milo 2004) is a formalism to model distributed systems represented as trees with function calls for tasks such as sending, receiving and querying data (like web services). Active XML function calls can be assimilated to NO-RL(3) rules that are applied in a forward-chaining manner to expand answers returned by XML queries. However, none of these existing work follows an OBDA approach, that is, they do not add a backward-chaining step of query reformulation on top of data management.

We leave as future work the precise analysis of QA combined complexity, the identification of tractable cases mixing NO-RL(2) and NO-RL(3) rules, as well as the design and evaluation of QA algorithms that would exploit the parallelization features of KV stores.

Acknowledgments. The work has been partially supported by projects PAGODA(ANR-12-JS02-007-01) and Labex PERSYVAL-Lab(ANR-11-LABX-0025-01).

References

Abiteboul, S.; Benjelloun, O.; and Milo, T. 2004. Positive Active XML. In *PODS*.

Abiteboul, S.; Hull, R.; and Vianu, V., eds. 1995. *Foundations of Databases: The Logical Level*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st edition.

Baget, J.-F.; Leclère, M.; Mugnier, M.-L.; and Salvat, E. 2011. On Rules with Existential Variables: Walking the Decidability Line. *Artificial Intelligence* 175(9-10):1620–1654.

Baumgartner, R.; Flesca, S.; Gottlob, G.; and Koch, C. 2001. Visual web information extraction with lixto. In *VLDB*.

Baumgartner, R.; Ceresna, M.; Gottlob, G.; Herzog, M.; and Zigo, V. 2003. Web information acquisition with lixto suite: a demonstration. In *ICDE*.

Calì, A.; Gottlob, G.; and Pieris, A. 2012. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence* 193:87–128.

Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *J. Autom. Reasoning* 39(3):385–429.

Eiter, T.; Ortiz, M.; Simkus, M.; Tran, T.; and Xiao, G. 2012. Query Rewriting for Horn-SHIQ Plus Rules. In *AAAI*.

Gottlob, G., and Koch, C. 2004. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM* 10.

Kifer, M.; Lausen, G.; and Wu, J. 1995. Logical foundations of object-oriented and frame-based languages. *J. ACM*.

Kifer, M. 2005. Rules and Ontologies in F-logic. In *RW*.

Kontchakov, R.; Lutz, C.; Toman, D.; Wolter, F.; and Zakharyaschev, M. 2010. The Combined Approach to Query Answering in DL-Lite. In *KR*.

Ong, K. W.; Papakonstantinou, Y.; and Vernoux, R. 2014. The SQL++ semi-structured data model and query language: A capabilities survey of sql-on-hadoop, nosql and newsql databases. *CoRR* abs/1405.3631.

Poggi, A.; Lembo, D.; Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Rosati, R. 2008. Linking data to ontologies. *J. Data Semantics* 10:133–173.

Sporny, M.; Longley, D.; Kellogg, G.; Lanthaler, M.; and Lindstrom, N. 2004. JSON-LD 1.0, A JSON-based Serialization for Linked Data. Technical Report http://www.w3.org/TR/json-ld/, W3C.