Mapping Data to Ontologies With Exceptions Using Answer Set Programming

Daniel P. Lupp and Evgenij Thorstensen

Abstract

In ontology-based data access (OBDA), databases are connected to an ontology via mappings from queries over the database to queries over the ontology. In this paper, we define an ASP-based semantics for mappings from relational databases to first-order ontologies, augmented with queries over the ontology in the mapping rule bodies. The resulting formalism can be described as "ASP modulo theories", and can be used to express constraints and exceptions in OBDA systems, as well as being a powerful mechanism for succinctly representing OBDA mappings. Furthermore, we show that brave reasoning in this setting has either the same data complexity as ASP, or is at least as hard as the complexity of checking entailment for the ontology queries. Moreover, despite the interaction of ASP rules and the ontology, most properties of ASP are preserved. Finally, we show that for ontologies with UCQ-rewritable queries there exists a natural reduction from our framework to ASP with existential variables.

1 Introduction

Ontology-based data access (OBDA) [25] is a method for data integration, utilizing a semantic layer consisting of an ontology and a set of mappings on top of a database. An ontology is a machine-readable model designed to faithfully represent knowledge of a domain independently of the structure of the database; it is comprised of concepts and relationships between these concepts. These ontologies are often formulated using description logics (DLs), a class of decidable logics, due to their desirable computational properties [6].

With the help of mappings, users' queries over the ontology are rewritten into a query over the database language, such as SQL, which can then be run on the source data. To ensure that this rewriting is always possible, one requires the ontology to be first-order rewritable (FOL-rewritable); that is, that every allowed query is equivalent to a first-order formula. However, not all description logics have this property. A common class of ontology languages used in OBDA is the DL-lite family. These description logics have been tailored towards FOL-rewritability and tractable query answering, making them ideally suited for OBDA [6].

Unfortunately, the rewriting step can cause a worst-case exponential blow-up in query size [6]. While this blow-up is necessary to ensure complete query answering, it can lead to highly redundant database queries. Robust pruning of redundant

This paper was presented at the NIK-2018 conference; see http://www.nik.no/.

queries without nonmonotonic features such as extensional constraints [27] or closed predicates [23] is practically infeasible. Furthermore, mapping design and maintenance is usually manual work [2]. This can be a very laborious task, and recent work on mapping evolution and repair [19] attempts to alleviate some of the difficulties involved. However, currently OBDA mappings are interpreted as first-order implications. As a consequence, they lack the expressivity to succinctly handle these issues: for instance, exceptions must be stated explicitly, possibly in multiple mapping assertions.

Example 1 *Let* `TABLE1(<ID>,<DATA>,<CONF>)` *be a table whose third column indicates a measure of confidentiality of the given entry, ranging from 1 (highly confidential) to 5 (not confidential). Furthermore, let* $A_C \sqsubseteq A$ *be an axiom in the ontology* $\mathcal{T}$*, where* A_C *represents the confidential individuals in* A*. The mapping assertions*

$$\forall X, Y \exists Z : \texttt{TABLE1}(X, Y, Z) \wedge Y = \text{“}a\text{”} \wedge (Z \neq \text{“}1\text{”} \wedge Z \neq \text{“}2\text{”} \wedge Z \neq \text{“}3\text{”}) \rightsquigarrow A(X)$$
$$\forall X, Y \exists Z : \texttt{TABLE1}(X, Y, Z) \wedge Y = \text{“}a\text{”} \wedge (Z = \text{“}1\text{”} \vee Z = \text{“}2\text{”} \vee Z = \text{“}3\text{”}) \rightsquigarrow A_C(X)$$

express that any entry whose `DATA` *column contains “a” is a member of the concept* A *or* A_C *depending on the confidentiality level. Changing what constistutes “confidential” to for instance “any data with confidentiality level 2 or above” can represent a major challenge for mapping maintenance: exceptions are listed explicitly and therefore must be changed in every relevant mapping assertion. Such code lists (exceptions depending on the value of a given column) are a common practice in database applications, yet are very prone to error since (1) changes must be made to potentially many mapping assertions and (2) it is entirely manual work, without robust consistency checking.*

Allowing for negation-as-failure as well as ontology queries in the mapping bodies alleviates these issues:the following mapping rules map the data in the same manner, yet have a more succicnt and robust manner of handling exceptions.

$$\forall X, Y \exists Z : \texttt{TABLE1}(X, Y, Z) \wedge Y = \text{“}a\text{”}, \textbf{not}\ A_C(X) \rightarrow A(X)$$
$$\forall X, Y \exists Z : \texttt{TABLE1}(X, Y, Z) \wedge Y = \text{“}a\text{”} \wedge (Z = \text{“}1\text{”} \vee Z = \text{“}2\text{”} \vee Z = \text{“}3\text{”}) \rightarrow A_C(X).$$

In this paper, we propose a new framework for OBDA mappings called mapping programs where mappings are not interpreted as first-order implications. Instead, they are rules containing a database query as well as (positive and negative) ontology queries in their bodies, allowing for existential quantification in both the body and the head of a rule. The ontology queries in rule bodies are evaluated with respect to both the answer sets of the mapping program and the ontology.

This integration of ontology queries into rules allows our formalism to express ontological epistemic constraints, for example extensional constraints [27] and thus a method of pruning redundant queries. Furthermore, by being able to express default rules, mapping programs serve as a powerful abbreviation tool for mapping maintenance (cf. Example 1). This enables the addition of nonmonotonic features to OBDA while retaining the desirable complexity of ontology reasoning. Furthermore, the semantics for OBDA with mapping programs is capable of capturing both the open-world reasoning of the ontology as well as the closed-world reasoning of the database. This was previously not possible with classical mappings, as they are interpreted as first-order theories and thus are inherently open-world.

Related work

Current research on extending OBDA with nonmonontonic capabilities has focused on the ontology side, e.g., through modal description logics or by inclusion of closed predicates [8, 23]. However, the modal semantics can be quite unintuitive. In this setting, modal ontology axioms do not behave well in the presence of nonmodal axioms. Furthermore, extending ontologies with closed predicates quickly results in intractability, cf. [23].

Using a rule-based framework for mappings in OBDA is no new notion; indeed, [3] considers mappings as Datalog programs (possibly with stratified negation) rather than first-order implications. However, this and to our knowledge all previously proposed mapping frameworks are monotonic and thus suffer from the issues illustrated in Example 1.

Since our goal is to connect data to an ontology, we require that each mapping rule contains a database query acting as a guard on the rule. Thus, existential witnesses generated by rules are not further propagated by the mapping program. This is in contrast to the more general existential rules frameworks of tuple-generating dependencies, where existentials in heads of rules may propagate [5, 4]. The decidability of mapping program reasoning therefore reduces entirely to decidability of ontology reasoning.

There have been several approaches to combining rule-based formalisms and description logic ontologies in contexts other than data transformation, be it by constructing a hybrid framework integrating both rules and ontology axioms into the same semantics [24] or by adding rules "on top" [10] of ontologies in the form of DL-programs. Here, rules can interact with an existing knowledge base by including special ontology queries in the rule bodies.

Both DL-programs and mapping programs are special cases of a more general framework called HEX programs [11]. These contain, in addition to regular atoms, queries to external sources in rule bodies which are evaluated with the help of oracles. While extending HEX programs with existential variables in the heads and negative bodies of rules, mapping programs restrict external source queries to database queries (in the positive body of rules, at least one such query must be present in each rule) and ontology queries (in the positive and negative body of rules). These restrictions guarantee that mapping programs are very well-behaved as opposed to general HEX-programs [9]: Despite existential quantification in rules, the presence of database queries ensures very limited and nonrecursive value invention (introduction of new constants), guaranteeing a finite grounding of every mapping program. This provides a solid foundation for query answering and data transformation, as it guarantees termination.

Paper overview

In the following, we define and analyze the general mapping program framework. We discuss the complexity of reasoning in our framework ($\mathrm{NP}^{\mathcal{O}}$-complete, if there exists an ontology reasoning oracle $\mathcal{O}$). Thus, despite the seemingly self-referential definition of satisfiability in mapping programs,[1] mapping program reasoning separates entirely, i.e., mapping programs can be described as "ASP modulo theories" [17] where for most reasoning tasks the ontology is treated as an external

[1] Entailment of rule bodies depends on both the ontology and the answer sets of the mapping program.

black box. As a result, many results from classical ASP, e.g., properties of stratified programs, are directly applicable to this setting. Finally, we consider a special case where the body ontology queries are UCQ-rewritable with respect to the ontology. Using this property, mapping programs can be reduced to classical ASP in a straightforward manner.

2 Preliminaries

OBDA Mappings

Let $\Sigma_{\mathcal{T}}$ and $\Sigma_{\mathcal{S}}$ be disjoint signatures containing *ontology predicate symbols*, and *source predicate symbols* respectively. Furthermore, let $\mathcal{C}$ be a set of constants. A *source schema* $\mathcal{S}$ is a relational schema containing relational predicates in $\Sigma_{\mathcal{S}}$ as well as integrity constraints. A *legal database instance* $\mathcal{D}$ over $\mathcal{S}$ is a set of ground atoms from $\Sigma_{\mathcal{S}}$ and $\mathcal{C}$ that satisfies all integrity constraints in $\mathcal{S}$. A first-order formula with free variables is called a *query*, if it has no free variables it is called a *boolean query*. An *ontology* $\mathcal{T}$ is a set of first-order formulas over $\Sigma_{\mathcal{T}}$. In practice, description logics are often used to express ontologies. Thus, though the results in this paper focus on the general case of FOL ontologies, we will use common DL notation throughout the examples in this paper for notational convenience [6].

Example 2 *The axiom* $Boss \sqsubseteq \exists hasSup^{-}$ *is equivalent to the first-order formula* $\forall x(Boss(x) \rightarrow \exists y.hasSup(y, x))$. *Here,* $hasSup^{-}$ *refers to the* inverse role *of* $hasSup$.

Following [18], an OBDA specification is a tuple $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ consisting of a database instance $\mathcal{D}$ legal over a schema $\mathcal{S}$, a FOL-rewritable ontology $\mathcal{T}$ and a set $\mathcal{M}$ consisting of *mapping assertions* of the form $m : \varphi \rightsquigarrow \psi$, where φ and ψ are queries over the data source and ontology, respectively. Then a model $\mathcal{I}$ of an OBDA specification $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ is a first-order model over $\Sigma_{\mathcal{T}} \cup \Sigma_{\mathcal{S}} \cup \mathcal{C}$ that satisfies both $\mathcal{T}$ and $\mathcal{M}$. Here we say that a first-order model $\mathcal{I}$ satisfies a mapping $\mathcal{M}$ if $\mathcal{I} \vDash \psi(\mathbf{t})$ for every mapping assertion $m : \varphi \rightsquigarrow \psi$ and every tuple $\mathbf{t} \in \text{eval}(\varphi, \mathcal{D})$.

Example 3 *Consider a database consisting of precisely one two-column table* `JOBS_DB(<NAME>,<JOB>)`. *Furthermore, consider the ontology* $\mathcal{T} = \{Empl \sqsubseteq Person, Boss \sqsubseteq Person\}$. *In the rewriting process, the query* $Person(x)$ *would be rewritten to* $Person(x) \sqcup Empl(x) \sqcup Boss(x)$ *while in the unfolding step, each of the above disjuncts would be expanded to a database query using the mapping assertions. For example, if there exist two mapping assertions* $\texttt{JOBS\_DB}(x, \text{“}Accountant\text{”}) \rightsquigarrow Empl(x)$ *and* $\texttt{JOBS\_DB}(x, \text{“}IT\text{”}) \rightsquigarrow Empl(x)$, *then the disjunct* $Empl(x)$ *would be unfolded as* $\texttt{JOBS\_DB}(x, \text{“}IT\text{”}) \vee \texttt{JOBS\_DB}(x, \text{“}Accountant\text{”})$.

Example 3 demonstrates some of the current shortcomings of classical OBDA mappings: due to its inherent, first-order nature, it is impossible to distinguish between inferred knowledge and knowledge that is explicit in the database. In the above example, in the presence of a mapping assertion $\texttt{JOBS\_DB}(x, y) \rightsquigarrow Person(x)$ the query $Person(x)$ would have sufficed without any ontology rewriting, since all desired information was contained in one table. However, while some OBDA implementations [16] support manual query pruning, i.e., the user is able to decide which concepts should not be rewritten, this can potentially lead to incomplete query answering, and there is currently no way of formally checking whether it

does. Thus, to ensure complete query answering we have a (potentially redundant) worst-case exponential blow-up in query size.

Another issue with the current aproach is how exceptions and a lack of information are dealt with. Currently, one must keep track of exceptions manually by explicitly listing all exceptions to a rule. Furthermore, due to the closed-world assumption (CWA) in the database, a lack of knowledge is interpreted as knowledge itself, e.g., if something is not contained in the `JOBS_DB` table, it is not a *Person*.

Answer Set Programming

Answer set programming (ASP) is a declarative programming paradigm based on the stable model semantics first defined in [14] as a means of handling default negation in a straightforward manner. It has become one of the more popular logic programming paradigms, due to, e.g., computational benefits such as guaranteed termination as compared to resolution in Prolog [20].

An *ASP-program* P s a set of rules of the form $H \leftarrow B_1, \ldots, B_m, \mathbf{not}\, C_1, \ldots, \mathbf{not}\, C_n.$ with ground atoms H, B_i, and C_j. The *head* of a rule r is $Head(r) = H$ and the *body* consists of a two parts, the negative body $body^-(r) = \{C_1, \ldots C_n\}$ and the positive body $body^+(r) = \{B_1, \ldots, B_m\}$. The Herbrand base HB_P of a program P is the set of all possible ground atoms using predicate symbols, function symbols and constants occuring in P. Then for a subset $I \subseteq HB_P$, the Gelfond-Lifschitz reduct P^I of P is the set of rules in P after applying the following changes: (1) If $C_i \in I$ for some i, remove the rule (this corresponds to rules that cannot be applied); (2) in all remaining rules, remove the negative clauses $\mathbf{not}\, C_i$ (this corresponds to removing all negative clauses that will evaluate to true).

This reduct is a program without any occurence of negation-as-failure. An interpretation $I \subseteq HB_P$ is called a *stable model* or an *answer set* of P if it is a $\subseteq$-minimal model of P^I, i.e., it is $\subseteq$-minimal and satisfies all rules in P^I.

Though the above semantics require ground atoms, i.e., are essentially propositional, ASP programs might also contains variables or function symbols. In this general case where function symbols are allowed, reasoning becomes undecidable [1]. In the function-free case, the first-order ASP programs are usually first grounded to reduce it to the propositional case. The grounded programs can then either be solved directly [13] or, e.g., translated to an instance of the Boolean satisfiability problem (SAT) before being passed on to efficient SAT solvers [22, 15].

3 OBDA Mapping Programs

In this section we introduce the syntax and semantics for a new framework for OBDA mappings called *mapping programs*. These programs consist of rules that, intuitively, map database queries $Q^{\mathcal{S}}$ to ontology queries $H^{\mathcal{T}}$ provided that certain conditions J^+ and J^- are met. Thus, mapping programs extend classical OBDA mappings with default reasoning.

A *mapping rule* is a rule of the form

$$H^{\mathcal{T}}(\mathbf{x}, \mathbf{z}) \leftarrow J_1^+(\mathbf{y}'_1), \ldots, J_l^+(\mathbf{y}'_l), \mathbf{not}\, J_1^-(\mathbf{y}_1), \ldots, \mathbf{not}\, J_k^-(\mathbf{y}_k), Q^{\mathcal{S}}(\mathbf{x}).$$

where $\mathbf{y}_i, \mathbf{y}'_j \subseteq \mathbf{x}$ for all i, j. Here, the *head* $H^{\mathcal{T}}(\mathbf{x}, \mathbf{z})$ is a first-order formula over $\Sigma_{\mathcal{T}}$ where $\mathbf{z}$ denotes possible existential variables. The *body* of a mapping rule consists of J_i^-, J_j^+, respectively called the *negative* and *positive justifications* and the *source*

query $Q^{\mathcal{S}}$. Here, J_i^- and J_j^+ are first-order formulas over the language of $\mathcal{T}$, and the *source query* $Q^{\mathcal{S}}$ is a first-order formula over $\Sigma_{\mathcal{S}}$. A set $\mathcal{M}$ of mapping rules is called a *mapping program*.

Example 4 *Consider a database consisting of one table* `Jobs_DB(<NAME>,<JOB>)`*. Let* $\Sigma_{\mathcal{T}} = \{Empl, hasSup, depHeadOf\}$ *with a unary relation* $Empl$ *of employees and two binary relations* $hasSup$ *and* $depHeadOf$*, describing a supervising relation and a department head relation, respectively. The default rule "employees, of whom we do not know that they are the head of a department, have a supervisor" can be expressed through the following mapping:*

$$m_1 : \exists Z.hasSup(X, Z) \leftarrow Empl(X), \mathbf{not}\, \exists Y.depHeadOf(X, Y), \texttt{Jobs\_DB}(X, P).$$

Then a *generalized OBDA specification* is a triple $(\mathcal{D}, \mathcal{M}, \mathcal{T})$, where $\mathcal{D}$ is a legal database instance over a schema $\mathcal{S}$, $\mathcal{M}$ is a mapping program, and $\mathcal{T}$ is an ontology.

Definition 1 (Skolem program, following [12]) *Let* $\mathcal{M}$ *be a mapping program. The* Skolem rule $sk(m)$ *associated to a rule* $m \in \mathcal{M}$ *is obtained by replacing each existential variable* v *in* $Head(m)$ *by a new Skolem function symbol* $sk_v(s)$*, where* s *is an ordered sequence of universal variables in* $Head(m)$ *. Then the* Skolem program *of* $\mathcal{M}$ *is* $sk(\mathcal{M}) = \{sk(m) \mid m \in \mathcal{M}\}$.

A *mapping interpretation* $\mathcal{A}$ is a consistent subset of $HB_{sk(\mathcal{M})}$, the Herbrand base over the Skolem program $sk(\mathcal{M})$. Such an interpretation is said to *satisfy* or *model* a positive Skolemized mapping rule

$$m : H^{\mathcal{T}}(\mathbf{x}, sk_{\mathbf{z}}(\mathbf{x})) \leftarrow J_1^+(\mathbf{y}_1'), \ldots, J_l^+(\mathbf{y}_l'), Q^{\mathcal{S}}(\mathbf{x}).$$

written $\mathcal{A} \vDash m$, if it satisfies the head or does not satisfy the body. It satisfies the body of a rule m if the following holds: for every tuple $\mathbf{t} \in \text{eval}(Q^{\mathcal{S}}, \mathcal{D})$, every interpretation I with $I \vDash \mathcal{T} \cup \mathcal{A}$ satisfies $J_j^+[\mathbf{t}]$ for all $j \leq l$. Here, $\text{eval}(Q^{\mathcal{S}}, \mathcal{D})$ denotes the set of tuples $\mathbf{t}$ that are answers to the query $Q^{\mathcal{S}}$ over $\mathcal{D}$.

Remark 1 *In this framework, the database query* $Q^{\mathcal{S}}$ *acts as a guard on the mapping rule* m*. It is in general a first-order query. Since* $Q^{\mathcal{S}}$ *is interpreted solely over* $\mathcal{D}$*, mapping rules are not applicable to existential witnesses generated by mapping rule heads. In particular, the database query* $\top(\mathbf{x})$ *is a shorthand for every tuple* $\mathbf{x}$ *occuring in the database.*

For brevity, we write $\mathcal{M}$ instead of $sk(\mathcal{M})$ by abuse of notation. Indeed, in the following we only consider the Skolemized mapping program.

An interpretation $\mathcal{A}$ is said to *satisfy* or *model* a positive mapping program $\mathcal{M}$, written $\mathcal{A} \vDash \mathcal{M}$, if it satisfies all mapping rules contained in $\mathcal{M}$.

Example 5 *Consider the mapping from Example 4. By Skolemizing, we get the mapping program:*

$$hasSup(X, sk_z(X)) \leftarrow Empl(X), \mathbf{not}\, \exists Y.depHeadOf(X, Y), \texttt{Jobs\_DB}(X, P).$$

Definition 2 (Partial ground program, following [12]) *The* partial grounding $PG(m)$ *of a mapping rule* m *is the set of all partial ground instances of* m *over constants in* $\Sigma_{\mathcal{D}}$ *for those variables that are not existential variables in the negative justifications. The* partial ground program *of a mapping program* $\mathcal{M}$ *is the set* $PG(\mathcal{M}) = \bigcup_{m \in \mathcal{M}} PG(m)$.

Example 6 *Consider the database and mapping from Examples 4 and 5. If the set of constants occuring in the database is $\{a, b\}$, then $PG(sk(m_1))$ consists of the four mapping rules for $u, v \in \{a, b\}$:*

$$hasSup(u, sk_z(u)) \leftarrow Empl(u), \mathbf{not}\ \exists Y.depHeadOf(u, Y), \texttt{Jobs\_DB}(u, v).$$

Remark 2 (Finite partial grounding) *Though the above definition is seemingly identical to that given in [12], the partial ground program of a mapping program is guaranteed to be finite due to the presence of the database guard Q^S in every mapping rule. This guard is evaluated solely over the domain of the database. In contrast, $\exists$-programs are partially ground using Skolem symbols as well, causing the partial ground program to be infinite as soon as any rule contains an existential head variable.*

Definition 3 ($\mathcal{T}$-reduct) *Given an ontology $\mathcal{T}$, define the $\mathcal{T}$-reduct $PG(\mathcal{M})^{\mathcal{A}}$ of a partial ground mapping program $PG(\mathcal{M})$ with respect to an interpretation $\mathcal{A}$ as the mapping program obtained from $PG(\mathcal{M})$ after applying the following:*

1. *Remove all mapping rules m where there exists some $i \leq k$ such that $\mathcal{T} \cup \mathcal{A} \models J_i^-$.*
2. *Remove negative justifications from the remaining rules.*

Example 7 *Continuing with our running example, let $\mathcal{T} = \{Boss \sqsubseteq \exists depHeadOf, Boss \sqsubseteq \exists hasSup^-\}$. Add the mapping rules $m_2 : Boss(X) \leftarrow \texttt{Jobs\_DB}(X, b)$ and $m_3 : Empl(X) \leftarrow \texttt{Jobs\_DB}(X, P)$. Then for $\mathcal{A} = \{\texttt{Jobs\_DB}(a, b), Empl(a), Boss(a)\}$, the rules*

$$hasSup(a, sk_z(a)) \leftarrow Empl(a), \mathbf{not}\ \exists Y.depHeadOf(a, Y), \texttt{Jobs\_DB}(a, v).$$

for $v \in \{a, b\}$ are removed in the $\mathcal{T}$-reduct $PG(\mathcal{M})^{\mathcal{A}}$ construction, since $\mathcal{T} \cup \mathcal{A} \models \exists Y.depHeadOf(a, Y)$. Then the $\mathcal{T}$-reduct w.r.t. $\mathcal{A}$ consists of all groundings of the following rules:

$$hasSup(b, sk_z(b)) \leftarrow Empl(b), \texttt{Jobs\_DB}(b, Y).$$
$$Boss(X) \leftarrow \texttt{Jobs\_DB}(X, b).$$
$$Empl(X) \leftarrow \texttt{Jobs\_DB}(X, P).$$

A mapping interpretation $\mathcal{A}$ is called a *$\mathcal{T}$-answer set* of $\mathcal{M}$ if it is a $\subseteq$-minimal model of the $\mathcal{T}$-reduct $PG(\mathcal{M})^{\mathcal{A}}$. Then a tuple $(\mathcal{I}, \mathcal{A})$ consisting of a first-order model $\mathcal{I}$ and a mapping interpretation $\mathcal{A}$ is a *model of a generalized OBDA specification* $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ if $\mathcal{I} \models \mathcal{T} \cup \mathcal{A}$ and $\mathcal{A}$ is a $\mathcal{T}$-answer set of $\mathcal{M}$. For a given ontology $\mathcal{T}$, a mapping program $\mathcal{M}$ is said to *entail a formula φ*, written $\mathcal{M} \models_{\mathcal{T}} \varphi$, if every $\mathcal{T}$-answer set of $\mathcal{M}$ entails φ. Similarly, a generalized OBDA specification $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ entails a formula φ, written $(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models \varphi$, if every model of $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ entails φ.

Example 8 *It is easily verifiable that the set $\mathcal{A}$ given in Example 7 is in fact a $\mathcal{T}$-answer set. It does not, however, entail $\mathcal{T}$, as the ontology axiom $Boss \sqsubseteq \exists hasSup^-$ is not satisfied. Thus, to obtain a model of the generalized OBDA specification, any model $\mathcal{I}$ must satisfy this axiom, in addition to the assertions in $\mathcal{A}$.*

Remark 3 (Extensional constraints) *Mapping programs are capable of expressing extensional constraints over the OBDA specification, i.e., constraints over the ontology language on the database and mappings [27]. For instance, the extensional constraint $C \sqsubseteq_e D$, which can be intuitively read as "if $C(a)$ is contained in the ABox, then $D(a)$ is contained in the ABox as well." Such a constraint is expressible with the mapping $\bot \leftarrow$ **not** $D(X), C(X), \top(X)$, where $\bot$ is* bottom *and $\top$ is the query* top *of appropriate arity. This guarantees that any $\mathcal{T}$-answer set of $\mathcal{M}$ must satisfy this constraint. Thus, queries containing the disjunction $C \sqcup D$ can be pruned, as querying for C in addition to D yields no new information. It is worth noting that, while this is similar to integrity constraints over the database, it is not entirely the same: the database schema might differ greatly from the structure of the ontology, thus allowing the possibility of describing database constraints on an ontology level.*

Complexity Analysis

In the general case, where the heads and bodies of mapping rules are allowed to contain arbitrary first-order formulas, reasoning over mapping programs is obviously undecidable. Indeed, consider an empty $\mathcal{T}$ and the mapping program $\mathcal{M} = \{R(a) \leftarrow \top, H(x) \leftarrow \varphi, R(x)\}$ for some arbitrary first-order formula φ. Then $\mathcal{M} \models H(a)$ if and only if φ is a tautology, which is known to be undecidable for arbitrary first-order φ. This is summarized in the following theorem.

Theorem 1 *The problem of checking $\mathcal{M} \models A$ for a given mapping program $\mathcal{M}$ and a ground atom A is undecidable.*

Corollary 1 *Let $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ be a generalized OBDA specification and A be a ground atom. Then the problem of checking $(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models A$ is undecidable.*

Now consider the case where $\mathcal{T} = \emptyset$ and $\mathcal{L}$ is the set of all ground atoms over the language of $\mathcal{T}$. In this case, the oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$ must only check membership in $\mathcal{A}$, hence it is linear in the size of $\mathcal{A}$. In this case, a partially ground Skolem mapping program is simply a classical ASP program. Therefore, brave reasoning over partially ground Skolemized mapping programs is at least as hard as classical ground ASP, i.e., is NP-hard [20].

More generally, let $(\mathcal{T}, \mathcal{L})$ be a pair consisting of an ontology $\mathcal{T}$ and a set $\mathcal{L}$ of formulas over the signature $\Sigma_\mathcal{T}$ such that $\mathcal{T}$-entailment of any $\varphi \in \mathcal{L}$ is decided by an oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$. In the following we consider mapping programs $\mathcal{M}$ where the queries in rules are formulas from $\mathcal{L}$. Then to construct a $\mathcal{T}$-answer set, we can employ a variant of the guess-and-check algorithm for ASP: By definition, a set $\mathcal{A}$ is a $\mathcal{T}$-answer set of $\mathcal{M}$ if and only if it is a $\subseteq$-minimal model of the $\mathcal{T}$-reduct $\mathcal{M}^\mathcal{A}$. Both the construction of $\mathcal{M}^\mathcal{A}$ and the satisfiability-checking are done following their respective definitions. For $\subseteq$-minimality, it suffices to check co-satisfiability of $\mathcal{A} \setminus \{a\}$ for every $a \in \mathcal{A}$, since $\mathcal{M}^\mathcal{A}$ is a positive program and hence monotonic.

The complexity of the guess-and-check method is dominated by the oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$: indeed, the oracles $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$ and co-$\mathcal{O}_{(\mathcal{T},\mathcal{L})}$ (the oracle that checks if a formula in $\mathcal{L}$ is *not* entailed by $\mathcal{T}$) are called a number of times polynomial in the size of $\mathcal{M}$.

More specifically, for a given oracle $\mathcal{O}_{(\mathcal{T},\mathcal{L})}$ brave reasoning over mapping programs is $\mathrm{NP}^{\mathcal{O}_{(\mathcal{T},\mathcal{L})}}$-complete.

Theorem 2 *Let $(\mathcal{T}, \mathcal{L})$ be a pair consisting of a first-order ontology $\mathcal{T}$ and a set of formulas $\mathcal{L}$ over the language of $\mathcal{T}$ such that $\mathcal{T}$-entailment is $|\mathcal{O}_{(\mathcal{T},\mathcal{L})}|$-hard for an oracle*

$\mathcal{O}_{(\mathcal{T},\mathcal{L})}$. *Then for a partially ground Skolemized mapping program $\mathcal{M}$ where the head and all justifications are formulas from $\mathcal{L}$, $\mathcal{T}$-answer set existence is NP$^{\mathcal{O}_{(\mathcal{T},\mathcal{L})}}$-complete.*

Note that, by the preceding theorem, a partially ground Skolemized mapping program satisfying the conditions of Theorem 2 can be rewritten into an ASP program with oracle calls in the rule bodies. Therefore, mapping programs can be considered as "ASP modulo theories." The resulting ASP program, however, bears little resemblance to the original program, as it is the encoding of an NP$^{\mathcal{O}_{(\mathcal{T},\mathcal{L})}}$ Turing machine.

Properties of Mapping Programs

In the previous section, we have seen that reasoning with mapping programs separates in the sense that a candidate $\mathcal{T}$-answer set $\mathcal{A}$ can be checked rule by rule using an ontology reasoning oracle. Therefore, many properties obtainable by syntactic restrictions on ASP transfer automatically to mapping programs. As an example, we show that the proof of answer set uniqueness of stratified ASP programs can be directly transferred to the mapping program setting.

Definition 4 *A mapping program $\mathcal{M}$ is said to be* stratified *if there exists a number l (called the* length of $\mathcal{M}$*) such that each query Q contained in a rule $\mathcal{M}$ can be associated to a natural number $v(Q) \leq l$ where for any rule r in $\mathcal{M}$ $v(Head(r)) \geq \max_{Q \in Body^+(r)} v(Q)$, and $v(Head(r)) > \max_{Q \in Body^-(r)} v(Q)$ hold.*

Then the proof of the following theorem is entirely analogous to its classical/DL-program counterparts [26, 10]. Indeed, using the iterative model sematics [26], a stratification gives rise to a perfect model $\mathcal{A}_{\mathcal{M}}$ of $\mathcal{M}$. It is then easily shown that this model must be a $\mathcal{T}$-answer set, and conversely that any $\mathcal{T}$-answer set is in turn a perfect model of $\mathcal{M}$.

Theorem 3 *Let $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ be a generalized OBDA specification with a stratified mapping program $\mathcal{M}$. Then $\mathcal{M}$ has a unique $\mathcal{T}$-answer set iff $\mathcal{M}$ is satisfiable.*

UCQ-Rewritable Justifications

We now analyze a restriction of mapping programs that admits a natural reduction to classical ASP for query answering and reasoning. To this end, let $\mathcal{T}$ be an ontology over a decidable fragment of first-order logic. We say a formula φ over $\Sigma_{\mathcal{T}}$ is *UCQ-rewritable with respect to $\mathcal{T}$* if the $\mathcal{T}$-rewriting of φ is equivalent to a union of conjunctive queries [7].

Then for a mapping program $\mathcal{M}$ where all justifications are UCQ-rewritable with respect to $\mathcal{T}$, let $\overline{\mathcal{M}}$, called the *$\mathcal{T}$-rewritten program*, denote the mapping program obtained from $\mathcal{M}$ by replacing every justification with its rewriting with respect to $\mathcal{T}$. The $\mathcal{T}$-rewritten program $\overline{\mathcal{M}}$ is equivalent to a program containing only atoms as positive justifications and CQs as negative justifications, by well-known logic program equivalence transformations [21]. By abuse of notation, $\overline{\mathcal{M}}$ will in the following denote this equivalent program.

Let us first establish the connection between mapping programs and $\exists$-ASP. Recall that a mapping rule can be applied to every tuple $\mathbf{t} \in \text{eval}(Q^{\mathcal{S}}, \mathcal{D})$ where $\mathcal{T} \cup \mathcal{A} \models J_i^+[\mathbf{t}]$ for all positive justifications J_j^+ and $\mathcal{T} \cup \mathcal{A} \not\models J_j^-[\mathbf{t}]$ for all negative

justifications J_j^-. If the TBox $\mathcal{T}$ is empty, this reduces to checking whether the justifications are certain answers w.r.t. $\mathcal{A}$ and hence simply checking containment in $\mathcal{A}$. This is, however, precisely the semantics of ASP with existential variables. Hence, mapping programs can be seen as an extension of $\exists$-ASP [12], both semantically and syntactically. This result is summarized in the following theorem.

Theorem 4 *Let $\mathcal{M}$ be a partially ground Skolem program where all justifications are conjunctive queries. Then a set $\mathcal{A}$ is a $\emptyset$-answer set of $\mathcal{M}$ iff it is a $\exists$-answer set of $\mathcal{M}$.*

The following lemma describes the relationship between $\mathcal{T}$-rewritten programs and reducts w.r.t. $\mathcal{A}$, which is particularly useful when analyzing the connection between $\exists$-ASP and mapping programs, as discussed in Theorem 5.

Lemma 1 *For any $\mathcal{A} \subseteq HB_{sk(\mathcal{M})}$ we have $\overline{\mathcal{M}}^{\mathcal{A}} = \overline{\mathcal{M}^{\mathcal{A}}}$, where $\overline{\mathcal{M}^{\mathcal{A}}}$ denotes the $\mathcal{T}$-rewritten program of $\mathcal{M}^{\mathcal{A}}$.*

Theorem 5 *Let $\mathcal{M}$ be a partially ground Skolem program where all justifications are UCQ-rewritable with respect to an ontology $\mathcal{T}$. A set $\mathcal{A}$ is a $\mathcal{T}$-answer set of $\mathcal{M}$ iff it is an $\emptyset$-answer set of $\overline{\mathcal{M}}$.*

As a direct consequence of the preceding theorem, the following corollary describes how query answering over an OBDA specification using a UCQ-rewritable mapping program can be reduced to query answering over an equivalent OBDA specification with an empty ontology.

Corollary 2 *Let $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ be an OBDA specification, $\overline{\mathcal{M}}$ the $\mathcal{T}$-rewritten program of $\mathcal{M}$, q a query over $\Sigma_{\mathcal{T}}$, and $\overline{q}$ its rewriting with respect to $\mathcal{T}$. Then $(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models q[t] \iff (\mathcal{D}, \overline{\mathcal{M}}, \emptyset) \models \overline{q}[t]$.*

Therefore, by Corollary 2 and Theorem 4 we find that every UCQ-rewritable mapping program $\mathcal{M}$ is equivalent (w.r.t. answer sets) to an $\exists$-ASP program. Then, by results in [12], this can be reduced to a classical ASP program. This is summarized in the following theorem.

Theorem 6 *For an OBDA specification $(\mathcal{D}, \mathcal{M}, \mathcal{T})$, where the justifications in $\mathcal{M}$ are UCQ-rewritable with respect to $\mathcal{T}$, there exists an ASP program $\mathcal{M}'$ such that for a query q over $\mathcal{T}$ $(\mathcal{D}, \mathcal{M}, \mathcal{T}) \models q[t] \iff \mathcal{M}' \models \overline{q}[t]$, i.e., query answering over $(\mathcal{D}, \mathcal{M}, \mathcal{T})$ reduces to cautious reasoning over $\mathcal{M}'$.*

4 Conclusion and Future Work

In this paper, we propose a new mapping framework for ontology-based data access (and data transformation in general) that is both very expressive and controllable with respect to computational complexity. Our framework allows for default reasoning as well as the expression of epistemic properties over the database and ontology. We show that mapping programs can be described as "ASP modulo theories," treating the ontology as an external black box. The loose coupling of ASP and ontological reasoning allows us to transfer existing results about ASP to this setting.

While various highly optimized ASP solvers do exist, the data complexity involved is rather undesirable in the context of real-world OBDA and big data.

Therefore, one of the greatest priorities regarding future work is to determine how and when the complexity can be reduced; the mapping program should not be run on the entire data set. Furthermore, there appears to be a strong connection between mapping programs and satisfiability modulo theories (SMT). We believe that mapping programs with ontologies supported by SMT solvers should be translatable into SMT instances in the same manner as ASP programs can be translated into SAT instances. If true, this could allow for very efficient query answering in certain OBDA settings. As such, investigating this connection is a clear goal for future work.

References

[1] M. Alviano, F. Calimeri, W. Faber, G. Ianni, and N. Leone. Function symbols in ASP: Overview and perspectives. In *NMR—Essays Celebrating Its 30th Anniversary*, pages 1–24. College Publications, 2011.

[2] N. Antonioli, F. Castan, S. Coletta, S. Grossi, D. Lembo, M. Lenzerini, A. Poggi, E. Virardi, and P. Castracane. Developing ontology-based data management for the italian public debt. In *22nd Italian Symposium on Advanced Database Systems, SEBD 2014*, pages 353–360. Universita Reggio Calabria and Centro di Competenza (ICT-SUD), 2014.

[3] R. Barilaro, N. Leone, F. Ricca, and G. Terracina. Distributed ontology based data access via logic programming. In *Proceedings of RR 2012*, pages 205–208, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res. (JAIR)*, 48:115–174, 2013.

[5] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

[6] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[7] F. Di Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *Proceedings of EDBT 2013*, pages 561–572, New York, NY, USA, 2013. ACM.

[8] F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *ACM Trans. Comput. Logic*, 3(2):177–225, April 2002.

[9] T. Eiter, M. Fink, T. Krennwallner, and C. Redl. Domain expansion for ASP-programs with external sources. *Artif. Intell.*, 233:84–121, 2016.

[10] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. *Art. Intel.*, 172(1213):1495 – 1539, 2008.

[11] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proceedings of IJCAI 2005*, pages 90–96, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[12] F. Garreau, L. Garcia, C. Lefèvre, and I. Stéphan. ∃-ASP. In *Proceedings of JOW 2015 co-located with the IJCAI 2015, Buenos Aires, Argentina, July 25-27,*, 2015.

[13] M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187-188:52–89, August 2012.

[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP '88*, pages 1070–1080. MIT Press, 1988.

[15] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability solvers. *Found. Art. Intel.*, 3:89–134, 2008.

[16] D. Hovland, D. Lanti, M. Rezk, and G. Xiao. Enabling SPARQL queries over enterprise relational data (extended version). *preprint*, 2015. arXiv:1605.04263v2 [cs.DB].

[17] J. Lee and Y. Meng. Answer set programming modulo theories and reasoning about continuous changes. In *Proceedings of the IJCAI 2013*, pages 990–996. AAAI Press, 2013.

[18] D. Lembo, J. Mora, R. Rosati, D. F. Savo, and E. Thorstensen. Mapping analysis in ontology-based data access: Algorithms and complexity. In *Proceedings of ISWC 2015*, pages 217–234, 2015.

[19] D. Lembo, R. Rosati, V. Santarelli, D. F. Savo, and E. Thorstensen. Approaching OBDA evolution through mapping repair. In *Proceedings of DL 2016.*, 2016.

[20] V. Lifschitz. What is answer set programming? In D. Fox and C. P. Gomes, editors, *Proceedings of AAAI 2008*, pages 1594–1597. AAAI Press, 2008.

[21] V. Lifschitz, L. R. Tang, and H. Turner. Nested expressions in logic programs. *Ann. Math. Art. Intel.*, 25(3):369–389, 1999.

[22] F. Lin and Y. Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Art. Intel.*, 157(12):115 – 137, 2004. Nonmonotonic Reasoning.

[23] C. Lutz, I. Seylan, and F. Wolter. Ontology-based data access with closed predicates is inherently intractable (sometimes). In *Proceedings of IJCAI 2013*, pages 1024–1030. AAAI Press, 2013.

[24] B. Motik and R. Rosati. Reconciling description logics and rules. *J. ACM*, 57(5):30:1–30:62, June 2010.

[25] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. In S. Spaccapietra, editor, *J. Data Sem. X*, pages 133–173. Springer-Verlag, Berlin, Heidelberg, 2008.

[26] T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[27] R. Rosati. Prexto: Query rewriting under extensional constraints in DLLite. In E. Simperl, P. Cimiano, A. Polleres, O. Corcho, and V. Presutti, editors, *The Semantic Web: Research and Applications*, volume 7295 of *Lecture Notes in Computer Science*, pages 360–374. Springer Berlin Heidelberg, 2012.