Existential Rule Languages with Finite Chase: Complexity and Expressiveness

Heng Zhang¹ and Yan Zhang¹ and Jia-Huai You²
¹School of Computing, Engineering and Mathematics, University of Western Sydney, Penrith, NSW 2751, Australia
²Department of Computing Science, University of Alberta, Edmonton, AB T6G 2E8, Canada

Abstract
Finite chase, or alternatively chase termination, is an important condition to ensure the decidability of existential rule languages. In the past few years, a number of rule languages with finite chase have been studied. In this work, we propose a novel approach for classifying the rule languages with finite chase. Using this approach, a family of decidable rule languages, which extend the existing languages with the finite chase property, are naturally defined. We then study the complexity of these languages. Although all of them are tractable for data complexity, we show that their combined complexity can be arbitrarily high. Furthermore, we prove that all the rule languages with finite chase that extend the weakly acyclic language are of the same expressiveness as the weakly acyclic one, while rule languages with higher combined complexity are in general more succinct than those with lower combined complexity.

Introduction
It has been shown that existential rule languages, also called Datalog끼, have prominent applications in ontological reasoning, knowledge representation, and databases, in which query answering is a primary yet challenging problem; see e.g., (Cali et al. 2010; Baget et al. 2011a). Under an existential rule language, queries are answered against a logical theory consisting of an input database and a finite set of existential rules, while a chase procedure is usually used. Specifically, given an input database \( D \), a finite set \( \Sigma \) of existential rules, and a query \( q \), we want to decide whether \( D \cup \Sigma \models q \).

Applying the chase procedure, the problem is equivalent to deciding whether \( \text{chase}(D, \Sigma) \models q \). Through a chase procedure, fresh nulls have to be introduced for each application of the existential rules, and hence, potential cyclic applications of these rules may lead the chase procedure to terminate, i.e., \( \text{chase}(D, \Sigma) \) is infinite. Therefore, the problem of query answering under existential rule languages is in general undecidable (Beeri and Vardi 1981).

There have been a considerable number of works on identifying decidable classes with respect to query answering. Basically, two major approaches have provided a landscape on this study: One is to focus on some restricted fragments of existential rule languages such that the underlying chase procedure, though non-terminating in general, still enjoys some kind of finite representability property, so that the problem of query answering is decidable under this setting. This paradigm includes, e.g., guarded rules (Cali, Gottlob, and Lukasiewicz 2012), greedy bounded treewidth sets (Baget et al. 2011b), sticky sets (Cali, Gottlob, and Pieris 2012), and Shy programs (Leone et al. 2012). The other approach is to identify a certain acyclicity condition under which each existential rule can only be finitely applied so that the chase procedure always terminates. There have been many recent studies on this paradigm. Our work presented in this paper is along this line. Below, let us provide a brief summary of recent works under this approach.

In their milestone paper, Fagin et al. (2005) formulated a concept called weak acyclicity (WA) as a sufficient condition to ensure the chase termination for existential rules. This concept was then extended to a number of notions, such as stratification (Deutsch, Nash, and Remmel 2008), super-weak acyclicity (Marnette 2009), local stratification (Greco, Spezzano, and Trubitsyna 2011), joint acyclicity (Krotzsch and Rudolph 2011), model-faithful acyclicity (MFA) and model-summarising acyclicity (MSA) (Grau et al. 2013), and some dependency relations by (Baget et al. 2014). Among these, MFA is known to define the largest rule class. In addition, many ontologies in various domains turn out to be in the MFA class, as evidenced in (Grau et al. 2013).

It has been observed that almost all of the existential rule languages defined based on the notion of acyclicity or its variations have PTIME-complete data complexity and 2-EXPTIME-complete combined complexity. The uniformity on data complexity is in fact due to an interesting result proved in (Marnette 2009), which states that every rule language with finite Skolem chase is in PTIME for data complexity. A natural question then arises: Does this uniformity hold for combined complexity? Moreover, what is the expressiveness of existing rule languages with finite chase? Please note that the uniformity on data complexity does not imply the uniformity on expressiveness as data complexity only captures the hardest case of a language.

Recently, there have been two interesting related works that studied the expressiveness of existential rules (Arenas, Gottlob, and Pieris 2014; Gottlob, Rudolph, and Simkus 2014). But both of them only focus on guarded language or its variations.
In this paper, we study the complexity, expressiveness and succinctness for existential rule languages with finite chase. Our contributions are summarized as follows:

1. A novel approach for classifying the existential rule languages with finite Skolem chase is proposed by restricting the use of existential variables in the Skolem chase. Under this approach, a family of interesting decidable rule languages, called bounded languages, are naturally defined. All of the existing rule languages with finite chase, e.g., the MFA class, are contained in these languages.

2. For every nonnegative integer k, the combined complexity of Boolean query answering for k-exponentially bounded language is proved to be (k + 2)-ExpTime-complete, and the membership problem of k-exponentially bounded language is proved to be in (k + 2)-ExpTime. Furthermore, for other bounded languages, the corresponding upper bounds of the complexity are also obtained.

3. All the languages with finite Skolem chase that extend the WA class are proved to be of the same expressiveness as WA, while languages with higher combined complexity are in general more succinct than those with lower combined complexity. On ordered databases, WA is shown to capture all existential rule sets whose universal models are computable in PTIME, even if they have no finite chase.

The results presented in this paper not only generalize some of the existing works, such as the two acyclicity notions of MFA and MSA proposed in (Grau et al. 2013), more importantly, they provide a global landscape for characterizing the existential rule languages with finite Skolem chase.

The rest of this paper is organized as follows. Section 2 provides necessary preliminaries. Section 3 defines a family of existential rule languages with finite Skolem chase called bounded classes, and presents some interesting properties of this family of languages. Section 4 then focuses on the complexity issues for bounded classes of languages, while section 5 explores the expressiveness and succinctness of these bounded classes of languages in details. Finally, section 6 concludes this paper with some remarks. Due to space limitation, proofs of some results are presented in an extended version of this paper, see (Zhang, Zhang, and You 2014).

Preliminaries

Databases and Queries. As usual, we assume (i) an infinite set $\Delta$ of constants, (ii) an infinite set $\Delta_n$ of (labelled) nulls, and (iii) an infinite set $\Delta_v$ of variables. A relational schema $\mathcal{R}$ consists of a finite set of relation symbols, each of which is armed with a natural number, its arity. Terms are either constants or variables. Every atomic formula (or atom) has the form $R(t)$ where $R$ is a relation symbol and $t$ a tuple of terms of a proper length. Ground terms are terms involving no variable, and facts are atoms built from ground terms.

Given a relational schema $\mathcal{R}$, an instance (database) over $\mathcal{R}$, or simply $\mathcal{R}$-instance ($\mathcal{R}$-database), is a (finite) set of facts involving only relation symbols from $\mathcal{R}$. The domain of a database $D$, denoted $\text{dom}(D)$, is the set of all constants appearing in $D$. General instances (databases) are the extensions of instances (databases) by allowing nulls to be used. Given a general instance (database) $D$ and a relational schema $\mathcal{R}$, the restriction of $D$ to $\mathcal{R}$, denoted $D|_{\mathcal{R}}$, is the set of facts in $D$ involving only relation symbols from $\mathcal{R}$.

A substitution is a function $h : \Delta \cup \Delta_n \cup \Delta_v \rightarrow \Delta \cup \Delta_n \cup \Delta_v$ with (i) $h(c) = c$ for all $c \in \Delta$ and (ii) $h(n) \in \Delta \cup \Delta_n$, for all $n \in \Delta_v$. Let $D$ and $D_0$ be general instances of the same schema. Then $D$ is called homomorphic to $D_0$, written $D \rightarrow D_0$, if there is a substitution $h$ with $h(D) \subseteq D_0$ where $h$ is assumed to be extended to atoms and general instances naturally. In this case, the function is called a homomorphism from $D$ to $D_0$. Moreover, $D$ is homomorphically equivalent to $D_0$ if $D$ is homomorphic to $D_0$ and vice versa.

Every conjunctive query (CQ) $q$ over a relational schema $\mathcal{R}$ has the form $q(x) := \exists y. \varphi(x, y)$, where $x$, $y$ are tuples of variables, and $\varphi(x, y)$ is a conjunction (sometimes we regard it as a set) of atoms with variables from $x$ and $y$, and relation symbols from $\mathcal{R}$. A Boolean CQ (BCQ) is a CQ of the form $q()$. Actually, BCQs can be regarded as general databases if we omit the quantifiers and regard the variables as nulls. Given any BCQ $q$ and any general instance $D$ over the same schema, the answer to $q$ over $D$ is “Yes”, written $D \models q$, if there exists a homomorphism from $q$ to $D$.

Existential Rules and Skolem Chase. Given a relational schema $\mathcal{R}$, every (existential) rule over $\mathcal{R}$ is a first-order sentence $\gamma$ of the form $\forall z \forall \gamma(\varphi(x, y) \rightarrow \exists z \psi(x, z))$, where $\varphi$ and $\psi$ are conjunctions of atoms with relation symbols from $\mathcal{R}$ and variables from $x \cup y$ and $x \cup z$, respectively. We call $\varphi$ the body of $\gamma$ and $\psi$ the head of $\gamma$, and write them as $\text{body}(\gamma)$ and $\text{head}(\gamma)$, respectively. When writing a rule, for simplicity, we will omit the universal quantifiers.

A rule ontology is a triple $(\Sigma, \mathcal{D}, \mathcal{Q})$, where $\Sigma$ is finite and nonempty set $\Sigma$ of rules, $\mathcal{D}$, called database schema, is a relational schema consisting of the relation symbols to be used in databases, and $\mathcal{Q}$, called query schema, is a relational schema consisting of the relation symbols to be used in queries. Relation symbols appearing in $\Sigma$ but neither $\mathcal{D}$ nor $\mathcal{Q}$ are called auxiliary symbols. Note that $\mathcal{D}$ and $\mathcal{Q}$ could be the same. Without loss of generality, in any rule ontology, each variable is assumed to be quantified at most once.

Let $\gamma$ be a rule $\varphi(x, y) \rightarrow \exists z \psi(x, z)$. We introduce a function symbol $f_z$ of arity $|x|$ for each variable $z \in \gamma$. From now on, we will regard terms built from constants and the introduced function symbols as a special class of nulls. The functional transformation of $\gamma$, denoted $\text{sk}(\gamma)$, is the formula obtained from $\gamma$ by substituting $f_z(x)$ for each variable $z \in \gamma$. Given a set $\Sigma$ of rules, the functional transformation of $\Sigma$, denoted $\text{sk}(\Sigma)$, is the set of rules $\text{sk}(\gamma)$ for all $\gamma \in \Sigma$.

Now we are in the position to define the (Skolem) chase. Let $D$ be a database and $\Sigma$ a rule set. We let $\text{chase}^n(D, \Sigma) = D$ and, for all $n > 0$, let $\text{chase}^{n+1}(D, \Sigma)$ denote the union of $\text{chase}^n(D, \Sigma)$ and $h(\text{head}(\gamma))$ for all rules $\gamma \in \text{sk}(\Sigma)$ and all substitutions $h$ such that $h(\text{body}(\gamma)) \subseteq \text{chase}^n(D, \Sigma)$. Let $\text{chase}(D, \Sigma)$ be the union of $\text{chase}^n(D, \Sigma)$ for all $n \geq 0$. It is well-known that, for all BCQs $q$, $D \uplus \Sigma \models q$ (under the semantics of first-order logic) if and only if $\text{chase}(D, \Sigma) \models q$. Given a rule ontology $\mathcal{O} = (\Sigma, \mathcal{D}, \mathcal{Q})$, we say that $\mathcal{O}$ has finite chase if for all $\mathcal{D}$-databases $D$, $\text{chase}(D, \Sigma)$ is finite. For more details, please refer to (Marnette 2009).
More Notations. Given a set \( \Sigma \) of rules and a \( \text{BCQ} \) \( q \), let \( |\Sigma| \) and \( |q| \) denote the numbers of symbols occurring in \( \Sigma \) and \( q \), respectively. We assume that the reader is familiar with complexity theory. Given a unary function \( T \) on natural numbers, by \( \text{DTIME}(T(n)) \) we mean the class of complexity languages decidable in time \( T(n) \) by a deterministic Turing machine. For \( k \geq 0 \) we let \( \exp_k(n) \) denote the function that maps \( n \) to \( n \) if \( k = 0 \), and \( 2^{\exp_{k-1}(n)} \) otherwise. By \( k\text{-EXP}\text{TIME} \) we mean the class \( \bigcup_{k>0} \text{DTIME}(\exp_k(n)) \).

For simplicity, we denote relation symbols (nulls/function symbols, respectively) by capitalized (lower-case, respectively) sans-serif letters, constants by lower-case italic letters \( a, b, c \), variables by lower-case italic letters \( u, v, w, x, y, z \), and terms by lower-case italic letters \( s, t \). All of these symbols may be written with subscripts or superscripts. In addition, bold italic letters \( u, v, w, x, y, z \) and \( s, t \) are used to range over tuples of variables and terms, respectively.

### Bounded Classes

In this section, we define a family of bounded rule languages with finite chase, and study its general properties.

We first define some notations. Given a ground term \( t \), the height of \( t \), denoted \( \text{ht}(t) \), is defined as follows:

\[
\text{ht}(t) := \begin{cases}
0 & \text{if } t \in \Delta; \\
\max\{\text{ht}(s) : s \in s\} + 1 & \text{if } t = f(s) \text{ for some } f.
\end{cases}
\]

Given any general instance \( A \), the height of \( A \), denoted \( \text{ht}(A) \), is defined as the maximum height of terms that have at least one occurrence in \( A \) if it exists, and \( \infty \) otherwise.

**Definition 1.** Every bound function is a function from positive integers to positive integers. Let \( \delta \) be a bound function. A rule ontology \( (\Sigma, D, Q) \) is called \( \delta \)-bounded if, for all \( D \)-databases \( D \), \( \text{ht}(\text{chase}(D, \Sigma)) \leq \delta(|\Sigma|) \). We let \( \delta\text{-BOUNDED} \) denote the class of \( \delta \)-bounded rule ontologies.

As there exist an infinite number of bound functions, it is interesting to know if there is a “maximum” bound function that captures all \( \delta \)-bounded rule ontologies for any bound function \( \delta \) (or all rule ontologies with finite chase). The following result shows that the answer is definitely “yes”.

**Proposition 1.** There is a bound function \( \delta \) such that, for every rule ontology \( O \), \( O \) has finite chase if it is \( \delta \)-bounded.

**Proof.** (Sketch) We first construct a bound function \( \delta \), and then it suffices to show that every rule ontology with finite chase is \( \delta \)-bounded. To define \( \delta \), we want to prove that, for every rule ontology \( O = (\Sigma, D, Q) \), there exists a database \( D_0 \) such that \( \text{ht}(\text{chase}(D_0, \Sigma)) \leq \text{ht}(\text{chase}(D_D, \Sigma)) \) for all \( D \)-databases \( D \). This can be done by employing the so-called critical database technique, which was developed in (Marneve 2009). Define \( \delta(n) \) the maximum \( \text{ht}(\text{chase}(D_0, \Sigma)) \) among all rule ontologies \( O = (\Sigma, D, Q) \) with finite chase such that \( |\Sigma| \leq n \); we then have the desired bound function \( \delta \).

**Remark 1.** Let \( \delta\text{-BOUNDED} \) be the union of \( \delta \)-BOUNDED for all bound functions \( \delta \). A rule ontology is called bounded if it belongs to \( \delta\text{-BOUNDED} \). As all bounded rule ontologies have finite chase, by Proposition 1 we have that \( \delta\text{-BOUNDED} \) contains exactly the rule ontologies with finite chase.

Next, let us define a class of interesting bound functions.

**Definition 2.** Let \( k \) be a natural number and let \( \exp_k \) be the function defined in the previous section. A rule ontology is called \( k\text{-exponentially bounded} \) if it is \( \exp_k \)-bounded.

**Remark 2.** The MFA class (Grau et al. 2013), which was shown to extend many existing languages with an acyclicity restriction, is defined by restricting the recursive uses of existential variables in Skolem chase. It is not difficult to see that \( \text{MFA} \subseteq \exp_0 \text{-BOUNDED} \). The following example shows that the inclusion is in fact strict.

**Example 1.** Let \( O = (\Sigma, D, Q) \) be a rule ontology, where \( D = \{R\} \) and \( \Sigma \) consists of the following rules:

\[
R(x, x) \rightarrow \exists y S(x, y) \land S(y, z) \\
R(x, y) \land S(z, x) \rightarrow \exists v R(z, v)
\]

This rule ontology does not belong to the MFA class because the existential variable \( v \) might be recursively applied in the Skolem chase (one can check it by letting the database \( D \) be \( \{R(a, a)\} \)). As each existential variable can be recursively used at most twice, \( O \) is \( 0 \)-exponentially bounded.

One might ask if all bounded ontologies can be captured by exponential bound functions (or computable bound functions). The proposition below shows that this is impossible.

**Proposition 2.** There is no computable bound function \( \delta \) such that every bounded rule ontology is \( \delta \)-bounded.

### Complexity

Now we study the complexity of bounded classes. We are interested in the complexity of two kinds of important computations: query answering and language membership.

**Boolean Query Answering.** The problem to be investigated here, also known as query entailment, is defined as follows: Given a set \( \Sigma \) of rules, a database \( D \) and a Boolean query \( q \), decide if \( D \cup \Sigma \models q \). We first consider the upper bound.

**Proposition 3.** Let \( \delta \) be a bound function. Then for any \( \delta \)-bounded rule ontology \( (\Sigma, D, Q) \), any \( D \)-database \( D \) and any \( \text{BCQ} \) \( q \) over \( Q \), it is in

\[
\text{DTIME}(\max(|\Sigma|, |\Sigma|^\delta(|\Sigma|))) \times \text{DTIME}(1)
\]

to check whether \( D \cup \Sigma \models q \).

**Proof.** (Sketch) First evaluate the size of \( \text{chase}(D, \Sigma) \). By this we know how many stages are needed for the chase to terminate. Counting the cost of each chase stage and querying on \( \text{chase}(D, \Sigma) \), we then have the desired result.

A lower bound for the combined complexity is as follows.

**Proposition 4.** It is \( (k+2)\text{-EXP}\text{TIME}\text{-hard} \) (for the combined complexity) to check, given a \( k \)-exponentially bounded rule ontology \( (\Sigma, D, Q) \), a \( D \)-database \( D \) and a \( \text{BCQ} \) \( q \) over \( Q \), whether \( D \cup \Sigma \models q \).

**Proof.** (Sketch) Let \( M \) be any deterministic Turing machine that terminates in \( \exp_{k+2}(n) \) steps on any input of length \( n \). Let \( D = \emptyset \) and \( Q = \{\text{Accept}\} \) where Accept is a nullary relation symbol. To show the desired result, it suffices to
show that, for each input (a binary string) \( x \), there is an exp\(_b\)-bounded rule ontology \( (\Sigma, D, Q) \) such that \( M \) terminates on input \( x \) if and only if \( \emptyset \cup \Sigma \models \text{Accept} \). Let \( x \) be an input of length \( n \). To construct the rule set \( \Sigma \), the main difficulty is to define a linear order of length \( \exp_b(k)(n) \). If the order is defined, by an encoding similar to that in (Dantsin et al. 2001), one can construct a set of datalog rules to encode both \( M \) and \( x \). Here we only explain how to define the linear order.

Let us first consider the case where \( k \) is even. The general idea is to construct a sequence of rule sets \( (\Sigma_i)_{i \geq 0} \). For each \( i \), let \( \text{Succ}_i, \text{Min}_i \), and \( \text{Max}_i \) be relation symbols intended to define the (immediate) successor relation, the minimum element and the maximum element, respectively, of a linear order. For \( i > 0 \), the function of \( \Sigma_i \) is as follows: If \( \text{Succ}_{i-1}, \text{Min}_{i-1} \) and \( \text{Max}_{i-1} \) define a linear order of length \( n \), then \( \text{Succ}_i, \text{Min}_i \), and \( \text{Max}_i \) define a linear order on integers (represented in binary strings) from 0 to \( 2^n \). To implement each \( \Sigma_i \), we generalize a technique used in the proof of Theorem 1 in (Calì, Gottlob, and Pieris 2010).

The first task is to define the binary strings of length one, i.e. “0” and “1”. This can be done by the following rule:

\[
\text{Min}_{i-1}(v) \rightarrow BS_1(v, 0) \land BS_1(v, 1)
\]

where \( BS_1(v, x) \) states that \( x \) is a binary string of length \( 2^i \).

The following rules are used to generate binary strings of length \( 2^{i+1} \) by combining two binary strings of length \( 2^i \):

\[
\begin{align*}
BS_1(v, x) \land BS_1(v, y) & \rightarrow \exists z \; C_1(v, x, y, z) \\
C_1(v, x, y, z) \land \text{Succ}_{i-1}(v, w) & \rightarrow BS_1(w, z) \\
\end{align*}
\]

Then, some rules to define a successor relation (w.r.t. the lexicographic order) on strings of length \( 2^{i+1} \) are as follows:

\[
\begin{align*}
C_1(v, x, y, z) \land C_1(v, x, y, z_0) \land \text{Succ}_i(v, y) \land \text{Succ}_{i-1}(v, w) & \rightarrow \text{Succ}_i(w, z, z_0) \\
C_1(v, x, y, z) \land C_1(v, x, y, z_0) \land \text{Max}_i(v, y) \land \text{Min}_{i-1}(v, y_0) & \rightarrow \text{Succ}_i(w, z, z_0) \\
\end{align*}
\]

where \( \text{Succ}_i(v, x, y) \) is intended to assert that \( y \) is the immediate successor of \( x \), and both \( x \) and \( y \) are of length \( 2^i \).

The minimum and the maximum binary strings of length \( 2^{i+1} \) are defined by the following rules:

\[
\begin{align*}
\text{Min}_i(v, x) \land \text{Min}_i(v, y) \land C_1(v, x, y, z) \land \text{Succ}_{i-1}(v, w) & \rightarrow \text{Min}_i(w, z) \\
\text{Max}_i(v, x) \land \text{Max}_i(v, y) \land C_1(v, x, y, z) \land \text{Succ}_{i-1}(v, w) & \rightarrow \text{Max}_i(w, z) \\
\end{align*}
\]

Now the desired relations \( \text{Num}_i, \text{Succ}_i, \text{Min}_i \), and \( \text{Max}_i \) can be obtained by applying the following rules:

\[
\begin{align*}
\text{Succ}_i(v, x, y) \land \text{Max}_{i-1}(v) & \rightarrow \text{Succ}_i(x, y) \\
\text{Min}_i(v, x) \land \text{Max}_{i-1}(v) & \rightarrow \text{Min}_i(x) \\
\text{Max}_i(v, x) \land \text{Max}_{i-1}(v) & \rightarrow \text{Max}_i(x) \\
\end{align*}
\]

For all \( i > 0 \), let \( \Sigma_i \) consist of all of the above rules. It is easy to see that \( \Sigma_i \) is as desired. Let 0, \ldots, \( n - 1 \) be distinct constants. Let \( \Sigma_0 \) denote the following rule set:

\[
\begin{align*}
\text{Min}_0(0) \land \text{Max}_0(n - 1) \\
\text{Succ}_0(0, 1) \land \cdots \land \text{Succ}_0(n - 2, n - 1) \\
\end{align*}
\]

Next, let \( \ell = k/2 + 1 \) and let \( \Sigma_{num} \) be the union of \( \Sigma_i \) for all \( i \) with \( 0 \leq i \leq \ell \). By the previous analysis, it is not difficult to see that \( \text{Succ}_i, \text{Min}_i \) and \( \text{Max}_i \) define a linear order on (the binary representations of) integers from 0 to \( \exp_b(k)(n) \). It is also not difficult to check that the rule ontology \( (\Sigma_{num}, \emptyset, \{\text{Accept}\}) \) is exp\(_b\)-bounded.

For the case where \( k \) is odd, we can achieve the goal by some slight modifications to \( \Sigma_{num} \): (i) substituting the least integer greater than or equal to \( \log n \) for \( n \) in \( \Sigma_0 \), and then (ii) setting \( \ell = k/2 + 2 \). Similarly, we can show that the resulting rule set \( \Sigma_{num} \) satisfies the desired property.

Now, by combining Propositions 3, 4, and the data complexity of Datalog (see, e.g., (Dantsin et al. 2001)), for any \( k \)-exponentially bounded class \( \mathcal{C} \), we then have the exact bound of the complexity w.r.t. Boolean query answering.

**Theorem 5.** For all integers \( k \geq 0 \), the Boolean query answering problem of the \( k \)-exponentially bounded language is \((k + 2)\text{-ExpTime}\)-complete for the combined complexity, and \( \text{PTIME}\)-complete for the data complexity.

**Membership of Language.** Now we consider the membership problem of bounded languages. The problem is as follows: Given a rule ontology, check whether it belongs to the bounded language under consideration. Since the boundedness is defined in a semantical way, it is interesting to know how to check whether a rule ontology is \( \delta \)-bounded.

**Proposition 6.** Let \( \delta \) be a bound function that is computable in \( \text{DTIME}(T(n)) \) for some function \( T(n) \). Then for every \( \delta \)-bounded rule ontology \( O = (\Sigma, D, Q) \), it is in

\[
\text{DTIME}\left( |\Sigma||\Sigma|^\Theta(\log |\Sigma|^{\Theta(1)}) + T(\log |\Sigma|^{\Theta(1)}) \right)
\]

to check whether \( O \) is \( \delta \)-bounded.

The above proposition can be proved by using Marnette’s critical database technique (2009) and then by an analysis similar to that in the proof of Proposition 3.

**Remark 3.** Two immediate corollaries of Proposition 6 are:

It is in \((k + 2)\text{-ExpTime}\) to check whether a rule ontology is \( k \)-exponentially bounded; moreover, the membership for \( \delta \)-bounded language is decidable whenever \( \delta \) is computable.

**Expressiveness and Succinctness.** Though all rule languages with finite chase are tractable for data complexity (Marnette 2009), in the last section we have shown that their combined complexity could be very high. Hence, a natural question is as follows: Are the languages with high combined complexity really necessary for representing ontological knowledge? In this section, we address this question on two aspects: What is the expressiveness of these languages? How about the succinctness among them?

**Universal Worldview Mapping.** We first propose a semantic (and more general) definition for rule ontologies.

**Definition 3.** Let \( D \) and \( Q \) be two relational schemas. A *universal worldview mapping*, or UWM for short, over \((D, Q)\) is a function that maps every \( D \)-database \( D \) to a general instance \( Q \) over \( Q \). Let \( \Phi \) and \( \Psi \) be two UWMs over \((D, Q)\). We say that \( \Phi \) is equivalent to \( \Psi \), written \( \Phi \approx \Psi \), if for all \( D \)-databases, \( \Phi(D) \) is homomorphically equivalent to \( \Psi(D) \).
It is clear that \( \approx \) is an equivalence relation on the UWMs. Next, we show how to define UWMs from rule ontologies.

**Definition 4.** Let \( O = (\Sigma, D, Q) \) be any rule ontology. We define \([O]\) as the function that maps every \( D \)-database \( D \) to the general instance chase\((D, \Sigma)\) if

Given any rule ontology \( O \), it is clear that \([O]\) is a UWM. We say that two rule ontologies \( O_1 \) and \( O_2 \) are equivalent if the corresponding UWMs are equivalent, i.e., \([O_1]\) \(\approx\) \([O_2]\). The following property explains why this is desired.

**Proposition 7.** Let \( O_1 = (\Sigma_1, D, Q) \) and \( O_2 = (\Sigma_2, D, Q) \) be two rule ontologies with finite chase. Then \([O_1] \approx [O_2]\) if, for all \( D \)-databases \( D \) and all \( BCQs \) \( q \) over \( Q \), we have

\[
D \cup \Sigma_1 \models q \iff D \cup \Sigma_2 \models q.
\]

In addition, for a technical reason, given a rule ontology \( O = (\Sigma, D, Q) \), we require that \( D \) and \( Q \) are disjoint and no relation symbol in \( D \) has an occurrence in the head of any rule in \( \Sigma \).\(^1\) We call such rule ontologies normal. These assumptions do not change the expressiveness since, for every relation symbol \( R \in D \cap Q \), we can always replace \( R \) in \( D \) with a fresh relation symbol \( R' \) of the same arity, and then add a copy rule \( R'(x) \rightarrow R(x) \) into the rule set \( \Sigma \).

**From Bounded Classes to the WA Class.** In this subsection, we show that any bounded ontology can be rewritten to a rule ontology that is weakly acyclic (Fagin et al. 2005).

Let us first review the notion of weak acyclicity. Fix \( \Sigma \) as a set of rules and \( R \) its schema. A position of \( \Sigma \) is a pair \((R, i)\) where \( R \in R \) is of an arity \( n \) and \( 1 \leq i \leq n \). The dependency graph of \( \Sigma \) is a directed graph with each position of \( \Sigma \) as a node, and with each pair \((R, i), (S, j)\) as an edge if there is a rule \( \varphi(x) \rightarrow \exists y \psi(x, y) \) from \( \Sigma \) such that either

- there is a variable \( x \in \varphi \) such that \( x \) occurs both in the position \((R, i)\) in \( \varphi \) and in the position \((S, j)\) in \( \psi \), or
- there are variables \( x \in \varphi \) and \( y \in \psi \) such that \( x \) occurs in the position \((R, i)\) in \( \varphi \) and \( y \) occurs in the position \((S, j)\) in \( \psi \) (in this case, the edge is called a special edge).

A rule ontology \( (\Sigma, D, Q) \) is weakly acyclic (WA) if no cycle in the dependency graph of \( \Sigma \) goes through a special edge.

It is well-known that the class of WA rule ontologies enjoys the finite chase property. In the last few years, a number of classes have been proposed to extend it. However, the next theorem shows that, in view of the expressiveness, the WA class is no weaker than any class with finite chase.

**Theorem 8.** For every normal rule ontology \( O = (\Sigma, D, Q) \) with finite chase, there exists a weakly acyclic normal rule ontology \( O^* = (\Sigma^*, D, Q) \) such that \([O] \approx [O^*]\).

We prove this theorem by developing a translation. The general idea is as follows. Given any normal rule ontology \( O = (\Sigma, D, Q) \) with finite chase, we need to construct a weakly acyclic rule ontology \( O^* = (\Sigma^*, D, Q) \) such that any computation on \( O \) can be simulated by a computation on \( O^* \). The main difficulty is how to simulate the cyclic existential quantifications by weakly acyclic existential quantifications. Fortunately, by Proposition 1, \( O \) is always bounded,

\[\tau(t) := \begin{cases} a\square \cdots \square & \text{if } t = a \in \Delta; \\ v^1 \cdots v^\ell \square \cdots \square & \text{if } t = v \in x; \\ f_i x_1^i \cdots x_\ell^i \square \cdots \square & \text{if } t = v \in z; \\ v^1 \cdots v^\ell & \text{if } t = v \in y. \end{cases}\]

where, in each of the first three cases, the tail of \( \tau(t) \) is filled with the symbol \( \square \) such that the length of \( \tau(t) \) is exactly \( \ell \).

In fact, we can use some relation symbol with a smaller arity, but this will make the presentation more complicated.

\(^1\)This is similar to that in data exchange (Fagin et al. 2005).
substituting $\tau(t)$ for each term $t \in \Delta \cup \Delta_v$, followed by
substituting $R^*$ for each relation symbol $R$.

In the chase procedure for the new ontology, by applying above rules on a $D$-database $D$, we obtain a fact set $S'$ that encodes chase $(D, \Sigma)$. Thus, as mentioned previously, the remaining task is to construct rules for the decoding. The idea is as follows: (i) let $\mathsf{Dom}^*$ be the set of all $\ell$-tuples that encode terms with occurrences in chase $(D, \Sigma)$; (ii) for each $\ell$-tuple $s^* \in \mathsf{Dom}^*$, generate a null $n$ for it (by applying an existential quantifier once), and use $\mathsf{Map}(s^*, n)$ to record the correspondence between $s^*$ and $n$; (iii) translate each fact $R^*(t^*)$ to a fact $R(t)$ by looking up the relation $\mathsf{Map}$.

To collect the $\ell$-tuples in stage (i), we need the following rules. Given an $n$-ary relation symbol $R \in Q$, let $\lambda_R$ denote
\[
\mathsf{Dom}^*((f_1, \ldots, f_n) \to \mathsf{Dom}^*(v_1) \land \cdots \land \mathsf{Dom}^*(v_n))
\]
where each $v_i$ is a tuple of distinct variables $v_i^1 \cdots v_i^r$, and $\mathsf{Dom}^*$ a fresh relation symbol of arity $\ell$.

Next, we define some rules to generate nulls, which implement stage (ii). For each function symbol $f_x$ in $\mathsf{sk}(\Sigma)$ where $x$ is an existential variable in $\Sigma$, let $\zeta_x$ denote
\[
\mathsf{Dom}^*((f_x, v) \to \exists x \mathsf{Map}(f_x, v, x))
\]
where $v$ is a tuple of distinct variables $v_1 \cdots v_{r-1}$, and $\mathsf{Map}$ a fresh $(\ell+1)$-ary relation symbol. In addition, let $\zeta_x$ denote
\[
\mathsf{Dom}^*((x, \square, \ldots, \square) \to \mathsf{Map}(x, \square, \ldots, \square, x))
\]
Informally, this rule asserts that, for any $\ell$-tuple that encodes a single-symbol term, we do not need to generate any null.

Now, we can define rules to carry out the decoding. For each $n$-ary relation symbol $R \in Q$, let $\vartheta_R$ denote
\[
\mathsf{Dom}^*(\mathsf{Dom}^*(v_1, \ldots, v_n) \land \mathsf{Map}(v_1, x_1) \land \cdots \land \mathsf{Map}(v_n, x_n)) \to R(x_1, \ldots, x_n).
\]
Finally, we let $\Sigma^*$ denote the rule set consisting of (1) $\vartheta_R$ for every relation symbol $R \in D$, (2) $\gamma^*$ for every rule $\gamma \in \Sigma$, (3) $\lambda_R$ for every relation symbol $R \in Q$, (4) $\zeta_x$ for every existential variable $x$ in $\Sigma$ such that $f_x$ is of a positive arity, (5) $\zeta_x$, and (6) $\vartheta_R$ for every relation symbol $R \in Q$.

**Example 2.** By adding a copy rule into the rule ontology $O$ defined in Example 1, we then obtain a normal rule ontology $O_0 = (\Sigma_0, D_0, Q_0)$, where $\Sigma_0$ is the following rule set:
\[
\begin{align*}
\mathsf{D}(x, y) & \to R(x, y) \\
R(x, x) & \to \exists y \mathsf{S}(x, y) \land \mathsf{S}(y, z) \\
R(x, y) & \land \mathsf{S}(x, z) \to \exists v \mathsf{R}(z, v)
\end{align*}
\]
and $D_0 = \{D\}$, $Q_0 = \{R\}$. Next, we will illustrate the translation by the rule ontology $O_0$.

All the function symbols in $\mathsf{sk}(\Sigma_0)$ are clearly unary, and as analyzed in Example 1, $O_0$ is $\delta$-bounded for some bound function $\delta$ with $\delta(|\Sigma_0|) = 2$. So, we have $\ell = 1^0 + 1^1 + 1^2 = 3$, i.e., terms generated by the chase procedure of $O_0$ will be encoded by triples of function-free terms. Now, we use the following rule to initialize the auxiliary relation symbol $D^*$:
\[
\mathsf{D}(x, y) \to D^*((x \square \square) y \square \square)
\]
To simulate the chase procedure of $O_0$, we need the following rules, which correspond to the rules in $\Sigma_0$:
\[
\begin{align*}
\mathsf{D}^*((x_1 x_2 x_3) y_1 y_2 y_3) & \to \mathsf{R}^*((x_1 x_2 x_3 y_1 y_2 y_3) \\
\mathsf{R}^*((x_1 x_2 \square x_1 x_2 \square) & \to \mathsf{S}^*((x_1 x_2 \square f_1 x_1 x_2) \land \mathsf{S}^*((f_1 x_1 x_2 f_2) x_1 x_2) \\
\mathsf{R}^*((x_1 x_2 x_3 y_1 y_2 y_3) & \land \mathsf{S}^*((y_1 y_2 y_3) z_1 z_2 \square) \to \mathsf{R}^*((z_1 z_2 \square f_1 z_1 z_2)
\end{align*}
\]
The following rules are used to implement the decoding:
\[
\begin{align*}
\mathsf{R}^*((x_1 x_2 x_3 y_1 y_2 y_3) & \to \mathsf{Dom}^*((x_1 x_2 x_3) \land \mathsf{Dom}^*((y_1 y_2 y_3) \\
\mathsf{Dom}^*(f_1 x_1 x_2) & \to \exists y \mathsf{Map}(f_1 x_1 x_2 y) \\
\mathsf{Dom}^*(f_2 x_1 x_2) & \to \exists z \mathsf{Map}(f_2 x_1 x_2 z) \\
\mathsf{Dom}^*(f_3 x_1 x_2) & \to \exists v \mathsf{Map}(f_3 x_1 x_2 v) \\
\mathsf{Dom}^*((x \square \square) & \to \mathsf{Map}(x \square \square, x) \\
\mathsf{R}^*((x_1 x_2 x_3 y_1 y_2 y_3) \land \mathsf{Map}(x \square \square, x) & \to \mathsf{R}(x y)
\end{align*}
\]
Finally, let $\Sigma_0^*$ consist of the set of all rules defined above. It is not difficult to check that $[O_0] \approx [\Sigma_0^*, D_0, Q_0]$. □

**Capturing PTIME by the WA Class.** We have proved that all the rule languages with finite chase are of the same expressiveness as the WA class in the last subsection. However, this characterization is syntactic. In this subsection, we will give a complexity-theoretic characterization. Before presenting the result, we need some definitions.

Like in traditional Datalog (Dantsin et al. 2001), we will study the expressiveness on ordered databases. Every ordered database $D$ is a database whose domain is an integer set $\{0, \ldots, n\}$ for some integer $n \geq 0$; whose schema contains three special relation symbols $\mathsf{Succ}$, $\mathsf{Min}$ and $\mathsf{Max}$ (we call such a schema ordered); in which $\mathsf{Succ}$ is interpreted as the immediate successor relation on natural numbers, and $\mathsf{Min}$ and $\mathsf{Max}$ are interpreted as $\{0\}$ and $\{n\}$, respectively. By ordered UWMs we mean the restrictions of UWMs to ordered databases. We generalize definitions of $[\cdot]$ and $\approx$ to ordered UWMs by replacing “database” with “ordered database”. Note that the ordered version of Proposition 7 still holds.

We fix a natural way for representing (general) databases in binary strings. Given a general database $D$, let $(D)$ denote its binary representation. Let $D$ and $Q$ be any two disjoint relational schemas where $D$ is ordered. Let $\Phi$ be an ordered UWM over $(D, Q)$. We say that $\Phi$ is computed by a Turing machine $M$ if $M$ halts on any input $(D)$ where $D$ is an ordered $D$-database, and there is a general $Q$-database $Q$ such that $Q$ is homomorphically equivalent to $\Phi(D)$ and the output w.r.t. $(D, Q)$ is $\langle Q \rangle$, the binary representation of $Q$.

On syntax, we also need a slightly richer language defined as follows. Let $D$ be a relational schema (as a database schema). A semipositive rule w.r.t. $D$ is a generalized rule defined by allowing negated atoms with relation symbols from $D$ to appear in the body. Semipositive rule ontologies are then generalized from rule ontologies by allowing semipositive rules w.r.t. its database schema. A semipositive rule ontology is called weakly acyclic if the rule ontology obtained by omitting negative atoms is weakly acyclic.

**Theorem 9.** For every ordered UWM $\Phi$ that is computable in deterministic polynomial time, there is a weakly acyclic and semipositive rule ontology $O$ such that $[\Phi] \approx [O]$. □
Remark 4. By a slight generalization of the critical database technique proposed in (Marnette 2009), one can show that every semipositive rule ontology with finite Skolem chase is computable in deterministic polynomial time. Therefore, the above theorem implies that every semipositive rule language with finite Skolem chase that extends the semipositive WA class captures the class of PTIME-computable UWMs.

Succinctness. Our previous results show that all the rule languages with finite chase that extend the weakly acyclic class are of the same expressiveness. Now we further consider the following question: Is it possible to compare the efficiency of rule languages with finite chase for representing ontological knowledge? In general, it is not an easy task to compare the succinctness for fragments of first-order logic. However, the following theorem provides us with such a result for rule languages, which states that the bounded rule languages with higher combined complexity are normally more succinct than those with lower combined complexity.

Theorem 10. For all $k > 0$, there exists a $k$-exponentially bounded rule ontology $O = (\Sigma, D, Q)$ such that, for any $(k - 1)$-exponentially bounded rule ontology $O_0 = (\Sigma_0, D, Q)$ where $\Sigma_0$ is of polynomial size w.r.t. $\Sigma$, we have $[O_0] \not\equiv [O]$.

Proof. (Sketch) Let $n, \ell$ and $\Sigma_{num}$ be defined as in the proof of Proposition 4. Let $D = \emptyset$ and $Q = \{\text{Min}, \text{Max}, \text{Succ}\}$. By using the notion of core (see, e.g., (Deutsch, Nash, and Remmel 2008)), we show a lower bound for the number of nulls in universal models. By estimating the number of nulls that are long as in the chase, then we prove that $(\Sigma_{num}, D, Q)$ is not equivalent to any $(k - 1)$-exponentially bounded ontology $(\Sigma, D, Q)$ if $\Sigma$ is of a polynomial size w.r.t. $\Sigma_{num}$.

Remark 5. Theorem 10 tells us that, although extending the WA class to larger classes with finite chase does not increase the expressiveness, the succinctness could be a bonus.

Remark 6. It would be interesting to compare the succinctness of finite-chase rule languages with the same combined complexity under query answering. For instance, is the MFA class more succinct than the WA class? But this is beyond the scope of this work. We will pursue it in the future.

Concluding Remarks

We have studied the existential rule languages with finite chase in this paper. Instead of considering specific rule languages like most current works on this topic, here we have defined a family of rule languages based on a new concept called $\delta$-boundedness, from which the overall complexity and expressiveness characterizations on these languages have been provided. Our study on this topic may be further undertaken in various directions. One interesting yet challenging future work is to investigate disjunctive existential rule languages. It is important to discover whether our approach can be extended to identify decidable disjunctive existential rule languages and to characterize relevant complexity and expressiveness properties. Results on this aspect may significantly enhance our current understanding on ontological reasoning with disjunctive existential rules (Alviano et al. 2012; Bourhis, Morak, and Pieris 2013).

References


