# Query Answering on Expressive Datalog± Ontologies

Mostafa Milani[1], Andrea Calì[2,3], and Leopoldo Bertossi[1]

[1]School of Computer Science
Carleton University, Canada

[2]Dept of Computer Science
Birkbeck, Univ. of London, UK

[3]Oxford-Man Institute of Quantitative Finance
University of Oxford, UK

{mmilani, bertossi}@scs.carleton.ca
andrea@dcs.bbk.ac.uk

## 1 Introduction

The Datalog$^{\pm}$ family of ontology languages [1], which extends Datalog with existential quantification, has been gaining importance in the area of Ontology Querying due to its capability of capturing several prominent Semantic Web languages as well as offering efficient query answering services in many variants relevant for applications. The core feature of Datalog$^{\pm}$ languages are so-called tuple-generating dependencies (TGDs), which are the main form of rules. Such rules allow the inference of new atoms from an initial set (a database), which is captured by the notion of *chase* procedure. For example, consider the TGD $r(X, Y) \rightarrow \exists Z\, s(X, Z)$ and a database $D$ constituted by a single atom $r(\mathsf{a}, \mathsf{b})$. A chase step will generate the atom $s(\mathsf{a}, \zeta)$, where $\zeta$ is a *labelled null*, that is, a placeholder for an unknown value; notice that the constant $\mathsf{b}$ is lost in this step as it doesn't appear in the new atom. Conjunctive query answering under general TGDs is undecidable; languages of the Datalog$^{\pm}$ family impose therefore restrictions on the form of rules so as to guarantee decidability and good computational properties. *Guarded* (extended by *weakly-guarded*) Datalog$^{\pm}$ were the first form of decidable Datalog$^{\pm}$ languages, inspired by guarded logic and characterised by the presence of a *guard atom* in each rule, that contains all variables of that rule.

The language called *sticky* Datalog$^{\pm}$ [1] was introduced in order to capture a "proper" notion of *join* in rules, that is, the occurrence of variables in two distinct atoms of a rule in the absence of a guard for that rule. *Weakly-sticky* Datalog$^{\pm}$ is an extension of sticky Datalog$^{\pm}$ that extends sticky Datalog$^{\pm}$ by also capturing the well-known class of *weakly-acyclic* TGDs.

Let $\Sigma$ be the following set of rules, where some body variables are *marked* (denoted by a hat sign, e.g. $\hat{X}$; see [1]) as the result of a procedure that identifies *positions* (arguments of predicates; for instance, $p[1]$ is the position corresponding to the first argument of the predicate $p$) where, in the chase procedure, values

can appear that can eventually be lost in a subsequent chase step.

$$v(X) \rightarrow \exists Y\ r(X,Y) \qquad r(X,\hat{Y}), r(\hat{Y},Z) \rightarrow p(X,Z).$$
$$p(\hat{X},\hat{Y}) \rightarrow \exists Z\ p(Y,Z) \qquad p(\hat{X},Y), p(Y,Z) \rightarrow t(Y,Z)$$

A set of rules is *sticky* if there is no marked variable in a rule that appears more than once in the body of the rule. Intuitively, stickiness can be defined by means of the following property: during a chase step according to a rule $\sigma$, each value corresponding to a variable appearing more than once in $\sigma$ is not lost in the chase step, and it is also never lost in any subsequent step involving atoms where it appears. $\Sigma$ is not sticky, as easily seen.

Weak-acyclicity is defined using the notion of rank of a position [2]. A position has finite (resp., infinite) rank if the number of labelled nulls that can appear in that position in the chase procedure is finite (infinite). In the Datalog$^{\pm}$ program $\Sigma$ above, $\Pi_F = \{v[1], r[1], r[2]\}$ is the set of positions with *finite rank* and $\Pi_\infty = \{p[1], p[2]\}$ is the set of positions with *infinite rank*. A Datalog$^{\pm}$ program is weakly-acyclic if all positions of predicates have finite rank. The program $\Sigma$ is not weakly-acyclic.

A set of rules is weakly-sticky if every marked variable that is repeated in the body of a rule appears at least once in a position with finite rank. This generalizes stickiness with acyclicity because the stickiness is only applied on values that appear in the positions with infinite rank. Here, $\Sigma$ is weakly-sticky. Specifically in the second rule, the repeated variable $Y$ appears in the positions $r[1]$ and $r[2]$ in $\Pi_F$. In the last rule, $Y$ is a repeated variable but not marked.

So far, no non-trivial deterministic algorithm for CQ answering has been devised for weakly-sticky Datalog$^{\pm}$. In this paper we set the basis for the implementation of conjunctive query (CQ) answering by the following contributions.

1. We propose a bottom-up technique, where we *ground* the rules of $\Sigma$ according to the database $D$ in a suitable way. The expansion instance terminates at a point dependent on the size of the query; the query can be then evaluated directly on the expanded instance.
2. We propose a *hybrid* approach to CQ answering as follows. First, we transform all positions in $\Pi_F$ into positions of rank 0 (that is, positions where only constants can appear); then certain variables in such positions are *grounded*, that is, the variables are replaced by selected constants of the initial instance. The obtained program is a sticky program, for which a well-known query rewriting technique for CQ answering can be applied.

## 2 A Grounding Approach

A first deterministic algorithm for query answering on a weakly-sticky Datalog$^{\pm}$ program $\Sigma$ and a database is obtained by grounding the rules of $\Sigma$ in a bottom-up fashion. The procedure is inspired by the version of the chase procedure in [5, 6]. We illustrate this technique by means of an example. Consider a database $D = \{p(\mathsf{a}, \mathsf{b}), v(\mathsf{b})\}$, a Boolean conjunctive query (BCQ) $q = \{u(X)\}$, and a set of WS rules $\Sigma$ as follows: $\sigma_1 : p(X,Y) \rightarrow \exists Z\ p(Y,Z)$ and $\sigma_2 : v(X), p(X,Y), p(Y,Z) \rightarrow u(Y)$.

We start from $D$ and iteratively generate ground rules by mapping via homomorphism the body of the rules in $\Sigma$ into $D$ or the head of the rules in $\Sigma'$ (the current set of ground rules). The basic procedure is as follows: we iteratively add a ground rule to $\Sigma'$ if its head atom is not homomorphic to the head of a rule already in $\Sigma'$, until no new rule can be added. This "cautious" procedure, similar to the chase procedure in [5, 6], guarantees its termination. In our example, the algorithm stops after adding only one ground rule $p(\mathsf{a}, \mathsf{b}) \rightarrow p(\mathsf{b}, \zeta_1)$ to $\Sigma'$, where $\zeta_1$ is a *labelled null*.

In order to complete our grounding, we *resume* the above basic procedure $M_q$ times, where $M_q$ is the number of existential variables in $q$. Before each resumption, we *freeze* the labelled nulls, which after having been frozen are considered as constants. In our example, we have only one more resumption, which adds the rules, $p(\mathsf{b}, \zeta_1) \rightarrow p(\zeta_1, \zeta_2)$ and $v(\mathsf{b}), p(\mathsf{b}, \zeta_1), p(\zeta_1, \zeta_2) \rightarrow u(\zeta_1)$. Resumptions are needed, intuitively, to capture applications of rules (in a chase procedure) where a join variable, appearing in two or more distinct atoms, are mapped to a labelled null.

Now, the number of resumptions depends on the query, which makes our grounding dependent on the query; however, for practical purposes, we could ground with $N$ resumptions so as to be able to answer queries with up to $N$ existential variables, and if a query with more than $N$ existential variables is to be answered, we can incrementally retake the already-computed grounding and add the required number of resumptions.

## 3 A Hybrid Approach

In this section, we propose an algorithm based on a hybrid approach that combines the grounding of specific variables in rules with the query rewriting algorithm from [3]. Given a set of WS rules $\Sigma$ and a database $D$, our algorithm transforms $\Sigma$ according to $D$ into a sticky set of rules $\Sigma'$. Sticky Datalog$^{\pm}$ enjoys *first-order rewritability*, that is, each CQ can be rewritten into a first-order query and directly answered on the database [3], providing the correct answer.

As an example, consider $D = \{v(\mathsf{a}), s(\mathsf{a}, \mathsf{b})\}$ and a set of WS rules $\Sigma$ that includes $\sigma_1$ and $\sigma_2$ from the example in Section 2 as well as the following rules:

$$\sigma_3 : v(X) \rightarrow \exists Y\ r(X, Y).$$
$$\sigma_4 : r(X, Y), s(X, Z) \rightarrow t(Y, Z).$$
$$\sigma_5 : r(X, Y), t(Y, Z) \rightarrow p(X, Z).$$

Let us call *weak rules* the rules in which some repeated marked body variables appear at least once in a position with finite rank; we call such variables *weak variables*. We aim at grounding the weak variables only, thus turning the WS program into a sticky program.

In our example, $\sigma_4$ and $\sigma_5$ are weak rules with $X$ and $Y$ as their weak variables respectively. $\sigma_4$ is partially grounded as a sticky rule $r(\mathsf{a}, Y), s(\mathsf{a}, Z) \rightarrow r(Y, Z)$ since $X$ can only take the value $a$ from $D$. To (partially) ground $\sigma_5$ we would need $Y$ to be replaced with a null value invented by $\sigma_3$. Notice that the variables

we are to ground can only be in positions with finite rank, which guarantees the finiteness of the partial grounding. Our algorithm consists of two phases.

*(Phase 1).* We remove the existential variables that are in the positions with finite rank with a method inspired by [4]. In the context of our example, we first Skolemize $\sigma_3$, that is the only rule with an existential variable in a position with finite rank. The result is $v(X) \rightarrow r(X, f(X))$. Next, we replace $r(X, f(X))$ with an expanded predicate $r'(X, \mathsf{f}, X)$ with the higher arity that results into a rule $v(X) \rightarrow r'(X, \mathsf{f}, X)$ in which $\mathsf{f}$ is a fresh constant that represents the replaced function. Then, we iteratively propagate the expanded predicate in the body and head of the rules, possibly expanding other predicates e.g. $t$, obtaining the new rules: $\sigma'_3 : v(X) \rightarrow r'(X, \mathsf{f}, X)$, $\sigma'_4 : r'(X, Y, Y'), s(X, Z) \rightarrow t'(Y, Y', Z)$, and $\sigma'_5 : r'(X, Y, Y'), t'(Y, Y', Z) \rightarrow p(X, Z)$. The result is a set of rules $\Sigma_{0,\infty} = \{\sigma_1, \sigma_2, \sigma'_3, \sigma'_4, \sigma'_5\}$ with positions of rank either zero or infinite.

*(Phase 2).* Now, we proceed with the grounding step where we replace the weak variables ($X$ in $\sigma'_4$ and $Y, Y'$ in $\sigma'_5$) with values from $D$ and the new constant $\mathsf{f}$. The result is a set of sticky rules $\Sigma'$ that includes $\sigma_1$, $\sigma_2$ and $\sigma'_3$ as well as the following rules, $\sigma''_4 : r'(\mathsf{a}, Y, Y'), s(\mathsf{a}, Z) \rightarrow t'(Y, Y', Z)$, and $\sigma''_5 : r'(X, \mathsf{f}, \mathsf{a}), t'(\mathsf{f}, \mathsf{a}, Z) \rightarrow p(X, Z)$.

To compute the answers to the query $q$, $\Sigma'$ and $q$ are rewritten into a first-order query using the rewriting method for sticky rules [3]. The first-order query is then evaluated on $D$.

## 4 Conclusion

Weakly-sticky Datalog$^{\pm}$ is an expressive ontology language with good computational properties and capable of capturing the most prominent Semantic Web languages. We proposed two deterministic algorithms for answering conjunctive queries on weakly-sticky Datalog$^{\pm}$. This, we believe, sets the basis for practical query answering algorithms for real-world scenarios. We plan to continue our work by running experiments on large data sets. We also intend to refine the hybrid algorithm by devising solutions to reduce the number of constants of the database used to ground the weak variables; this would improve the efficiency of our approach.

## References


1. A. Cali, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence*, 2012, 193:87-128.
2. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 2005, 336:89-124.
3. G. Gottlob, G. Orsi, and A. Pieris. Query Rewriting and Optimization for Ontological Databases. *Proc. TODS*, 2014, 39(3):25.
4. Krötzsch, M. and Rudolph, S. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. *Proc. IJCAI*, 2011, pp. 963-968.
5. N. Leone, M. Manna, G. Terracina, and P. Veltri. Efficiently Computable Datalog$^{\pm}$ Programs. *Proc. KR*, 2012, pp. 13-23.
6. M. Milani, L. Bertossi. Tractable Query Answering and Optimization for Extensions of Weakly-Sticky Datalog+-. *Proc. AMW*, 2015.