The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Goal-Driven Query Answering for Existential Rules with Equality

Michael Benedikt, Boris Motik
University of Oxford

Efthymia Tsamoura
Alan Turing Institute & University of Oxford

Abstract

Inspired by the magic sets for Datalog, we present a novel goal-driven approach for answering queries over terminating existential rules with equality (aka TGDs and EGDs). Our technique improves the performance of query answering by pruning the consequences that are not relevant for the query. This is challenging in our setting because equalities can potentially affect all predicates in a dataset. We address this problem by combining the existing singularization technique with two new ingredients: an algorithm for identifying the rules relevant to a query and a new magic sets algorithm. We show empirically that our technique can significantly improve the performance of query answering, and that it can mean the difference between answering a query in a few seconds or not being able to process the query at all.

1 Introduction

Existential rules with equality, also known as *tuple- and equality generating dependencies* (TGDs and EGDs) or Datalog$^{\pm}$ rules, extend Datalog by allowing rule heads to contain existential quantifiers and the equality predicate $\approx$. Answering a conjunctive query Q over a set of existential rules Σ and a base instance B is key to dealing with incomplete information in information systems (Fagin et al. 2005). The problem is undecidable in general, but many decidable cases are known (Baget et al. 2011b; König et al. 2015; Baget et al. 2015a; Gottlob, Manna, and Pieris 2015; Leone et al. 2012). Systems such as Llunatic (Geerts et al. 2014), RDFox (Motik et al. 2014), DLV$^{\exists}$ (Leone et al. 2012), ChaseFUN (Bonifati, Ileana, and Linardi 2017), Ontop (Calvanese et al. 2017), and Graal (Baget et al. 2015b) implement various query answering techniques. One solution to this problem is to evaluate the query in a *universal model* of $\Sigma \cup B$, and a common and practically relevant case is when a finite universal model can be computed using a *chase* procedure. Many chase variants have been proposed. Although checking chase termination is undecidable for all variants (Deutsch, Nash, and Remmel 2008; Marnette 2009), numerous sufficient *acyclicity* conditions (Marnette 2009; Krötzsch and Rudolph 2011; Grau et al. 2013) guarantee termination of at least the oblivious Skolem chase; we call such $\Sigma \cup B$ *terminating*.

Computing a universal model in full when only a specific query is to be answered may be inefficient. We experimentally show that query answers often depend only on a small subset of the universal model, particularly for queries containing constants, so the chase may perform a lot of unnecessary work. Moreover, universal models sometimes cannot be computed due to their size. In such cases, *goal-driven* query answering techniques, which take the query into account, hold the key to efficient query answering.

One possibility, implemented in systems such as Ontop (Calvanese et al. 2017) and Graal (Baget et al. 2015b), is to *rewrite* the relevant rules into a new query that can be evaluated directly on the base instance. Rewriting into first-order queries is possible for DL-lite (Calvanese et al. 2007), linear TGDs (Calì, Gottlob, and Lukasiewicz 2012), and sticky TGDs (Gottlob, Orsi, and Pieris 2014; Calì, Gottlob, and Pieris 2010), among others. Frontier-guarded (Baget et al. 2011a), weakly-guarded (Gottlob, Rudolph, and Simkus 2014), and Horn-$\mathcal{SHIQ}$ (Eiter et al. 2012) rules can be rewritten into Datalog. Rewriting approaches, however, cannot handle common properties that can be handled via the chase, such as transitivity, and they typically support only "innocuous" equalities that do not affect query answers.

In Datalog and logic programming, the *magic sets* algorithm (Bancilhon et al. 1986; Beeri and Ramakrishnan 1991) annotates the rules with *magic* atoms, which ensure that bottom-up evaluation of the magic program simulates top-down query evaluation. This influential idea has been adapted to disjunctive (Alviano et al. 2012a) and finitely recursive (Calimeri et al. 2009) programs, programs with aggregates (Alviano, Greco, and Leone 2011), and *Shy* existential rules (Alviano et al. 2012b). These approaches, however, do not handle existential rules with equality.

In this paper we present what we believe to be the first goal-driven query answering technique for terminating existential rules *with* equality. Given a set of rules Σ and a query Q, we compute a logic program P such that, for each base instance B, the answers to Q on $\Sigma \cup B$ and $P \cup B$ coincide, but processing the latter is typically much more efficient. Our approach combines existing techniques such as *singularization* (Marnette 2009) with two new ingredients: a new *relevance analysis* algorithm that identifies irrelevant rules, and a new magic sets variant that handles existential rules with equality. These two techniques are complementary: the first one prunes rules whose consequences are irrelevant to the query, and the second one prunes the irrelevant consequences of the remaining rules. Since equalities can potentially affect any predicate, both techniques are needed to efficiently identify the relevant equalities.

We have empirically evaluated our technique on a recent benchmark that includes a diverse set of existential rules (Benedikt et al. 2017). Our results show that goal-driven query answering is generally more efficient than computing the chase in full. In fact, our approach can mean the difference between success and failure: even though the chase cannot be computed in several cases, we can answer the relevant queries in a few seconds. We also show that relevance analysis alone is very effective at eliminating irrelevant rules even without equalities. Finally, we show that magic sets alone can be less efficient on queries without constants, but it greatly benefits queries with constants. A combination of both techniques usually provides the best performance.

All proofs and further experimental results are given in the extended version (Benedikt, Motik, and Tsamoura 2017).

2 Preliminaries

We use the standard first-order logic notions of *variables*, *constants*, *function symbols*, *predicates*, *arity*, *terms*, *atoms*, and *formulas*, and $\approx$ is the binary *equality* predicate. Atoms $\approx(s,t)$ are *equational* and are usually written as $s \approx t$, and all other atoms are *relational*. A *fact* is a variable-free atom, an *instance* is a (possibly infinite) set of facts, and a *base instance* is a finite, function-free instance. We consider two notions of entailment: $\models$ interprets $\approx$ as an "ordinary predicate" without any special semantics, whereas $\models_\approx$ interprets $\approx$ under the usual semantics of equality without the *unique name assumption* (UNA)—that is, distinct constants can be derived equal. A theory T *satisfies UNA* if no two distinct constants a and b exist such that $T \models_\approx a \approx b$. For example, let $\varphi = A(a) \wedge a \approx b$. Then, $\varphi \models_\approx A(b)$ and $\varphi \models_\approx b \approx a$, and φ does not satisfy UNA. In contrast, $\varphi \not\models A(b)$ and $\varphi \not\models b \approx a$. We often abbreviate a tuple $t_1, \ldots, t_n$ as $\mathbf{t}$, and we often treat $\mathbf{t}$ as a set and write $t_i \in \mathbf{t}$.

A term t *occurs* in a term, atom, tuple, or set X if X contains t possibly nested inside another term; $\mathsf{vars}(X)$ is the set of variables occurring in X; and X is *ground* if $\mathsf{vars}(X) = \emptyset$. For σ a mapping of variables and/or constants to terms, $\sigma(X)$ replaces each occurrence of a term t in X with $\sigma(t)$ if the latter is defined, and σ is a *substitution* if its domain is finite and contains only variables. For μ a mapping of ground terms to ground terms, $\mu[\![X]\!]$ replaces each occurrence of a term t not nested in a function symbol with $\mu(t)$ if the latter is defined. For example, let $A = R(f(x), g(a))$; then, $\sigma(A) = R(f(b), g(c))$ for $\sigma = \{x \mapsto b, a \mapsto c\}$, and $\mu[\![A]\!] = R(f(x), h(d))$ for $\mu = \{a \mapsto b, g(a) \mapsto h(d)\}$.

Existential rules are logical implications of two forms: $\forall \mathbf{x}.[\lambda(\mathbf{x}) \rightarrow \exists \mathbf{y}.\rho(\mathbf{x},\mathbf{y})]$ is a *tuple-generating dependency* (TGD), and $\forall \mathbf{x}.[\lambda(\mathbf{x}) \rightarrow t_1 \approx t_2]$ is an *equality-generating dependency* (EGD), where $\lambda(\mathbf{x})$ and $\rho(\mathbf{x},\mathbf{y})$ are conjunctions of relational, function-free atoms with variables in $\mathbf{x}$ and $\mathbf{x} \cup \mathbf{y}$, respectively, t_1 and t_2 are variables from $\mathbf{x}$ or constants, and each variable in $\mathbf{x}$ occurs in $\lambda(\mathbf{x})$. Quantifiers $\forall \mathbf{x}$ are commonly omitted. Conjunction $\lambda(\mathbf{x})$ is the *body* of a rule, and $\rho(\mathbf{x},\mathbf{y})$ and $t_1 \approx t_2$ are its *head*. We assume that queries are defined using a *query predicate* Q that does not occur in rule bodies or under existential quantifiers. A tuple $\mathbf{a}$ of constants is an *answer* to Q on a finite set of existential rules Σ and a base instance B iff $\Sigma \cup B \models_\approx \mathsf{Q}(\mathbf{a})$.

When treating $\approx$ as "ordinary," we allow rule bodies to contain equality atoms, and we can axiomatize the "true" semantics of $\approx$ for Σ as follows. Let $\mathsf{R}(\Sigma)$ and $\mathsf{C}(\Sigma)$ contain the *reflexivity* (1) and *congruence* (2) axioms, respectively, instantiated for each n-ary predicate R in Σ distinct from $\approx$ and each $1 \leq i \leq n$. Let ST contain the *symmetry* (3) and the *transitivity* (4) axioms. We assume that each base instance contains only the predicates of Σ, since the equality axioms are then determined only by Σ. Then, for each base instance B and tuple of constants $\mathbf{a}$, we have $\Sigma \cup B \models_\approx \mathsf{Q}(\mathbf{a})$ if and only if $\Sigma \cup \mathsf{R}(\Sigma) \cup \mathsf{C}(\Sigma) \cup \mathsf{ST} \cup B \models \mathsf{Q}(\mathbf{a})$.

$$R(\ldots, x_i, \ldots) \rightarrow x_i \approx x_i \tag{1}$$

$$R(\ldots, x_i, \ldots) \wedge x_i \approx x_i' \rightarrow R(\ldots, x_i', \ldots) \tag{2}$$

$$y \approx x \rightarrow x \approx y \tag{3}$$

$$x \approx y \wedge y \approx z \rightarrow x \approx z \tag{4}$$

Our algorithms use logic programming, which we define next. A *rule* r has the form $R(\mathbf{t}) \leftarrow R_1(\mathbf{t}_1) \wedge \cdots \wedge R_n(\mathbf{t}_n)$, where $R(\mathbf{t})$ and $R_i(\mathbf{t}_i)$ are atoms possibly containing function symbols. Each variable in r must occur in some $\mathbf{t}_i$. To distinguish existential from logic programming rules, we use $\rightarrow$ for the former and $\leftarrow$ for the latter. Conjunction $\mathsf{b}(r) = R_1(\mathbf{t}_1) \wedge \cdots \wedge R_n(\mathbf{t}_n)$ is the *body* of r and we often treat it as a set, and atom $\mathsf{h}(r) = R(\mathbf{t})$ is the *head* of r. Predicate $\approx$ is always ordinary in logic programming, so R and R_i can be $\approx$. A *(logic) program* P is a finite set of rules, and it is interpreted in first-order logic as usual. Again, we assume that a query in P is defined using the predicate Q not occurring in rule bodies. For I an instance, $T_P(I)$ is the result of extending I with $\sigma(\mathsf{h}(r))$ for each rule $r \in P$ and substitution σ such that $\sigma(\mathsf{b}(r)) \subseteq I$. Finally, for B a base instance, we inductively define a sequence of interpretations where $I_0 = B$ and $I_i = T_P(I_{i-1})$ for $i > 0$; then, the *least fixpoint* of P on B is $T_P^\infty(B) = \bigcup_{i \geq 0} I_i$. It is well known that $P \cup B \models F$ iff $F \in T_P^\infty(B)$ holds for each fact F.

Our algorithms reduce query answering over existential rules to reasoning in logic programming. We eliminate existential quantifiers by computing the *Skolemization* $\mathsf{sk}(\Sigma)$ of a set Σ of existential rules. Set $\mathsf{sk}(\Sigma)$ contains each EGD of Σ as a logic programming rule and, for each TGD $\tau = \lambda(\mathbf{x}) \rightarrow \exists \mathbf{y}.\rho(\mathbf{x},\mathbf{y}) \in \Sigma$ and each $R(\mathbf{t}) \in \rho(\mathbf{x},\mathbf{y})$, set $\mathsf{sk}(\Sigma)$ contains the rule $\sigma(R(\mathbf{t})) \leftarrow \lambda(\mathbf{x})$ where σ is a substitution mapping each variable $y \in \mathbf{y}$ to $f_{\tau,y}(\mathbf{x}')$ for $\mathbf{x}' = \mathsf{vars}(\lambda(\mathbf{x})) \cap \mathsf{vars}(\rho(\mathbf{x},\mathbf{y}))$ and $f_{\tau,y}$ a fresh function symbol unique for τ and y. Let $P = \mathsf{sk}(\Sigma)$; if Σ and B do not contain $\approx$, then for each predicate R and tuple $\mathbf{a}$ of constants, we have $\Sigma \cup B \models R(\mathbf{a})$ iff $P \cup B \models R(\mathbf{a})$. If Σ or B contains $\approx$, we can axiomatize equality using axioms $\mathsf{R}(P)$, $\mathsf{C}(P)$, and ST defined analogously to (1)–(4); then, $P' = P \cup \mathsf{R}(P) \cup \mathsf{C}(P) \cup \mathsf{ST}$ captures the intended semantics of $\approx$, and $\Sigma \cup B \models_\approx R(\mathbf{a})$ iff $P' \cup B \models R(\mathbf{a})$.

Now let P and P' be as in the previous paragraph. We could answer queries over such P' by computing $T_{P'}^\infty(B)$

and evaluating Q on it, but this is inefficient even when P is just a Datalog program (Motik et al. 2015) since firing congruence rules can be prohibitively expensive. The *chase for logic programs* offers a more efficient method for reasoning with $P' \cup B$ by efficiently computing a representation of $T_{P'}^{\infty}(B)$. It is applicable if P does not contain constants, function symbols, or $\approx$ in the rule bodies. The algorithm constructs a sequence of pairs $\langle I_i, \mu_i \rangle$, $i \geq 0$, where I_i is an instance and μ_i maps ground terms to ground terms. The algorithm initializes I_0 to a normalized version of B where all constants reachable by $\approx$ in B are replaced by a representative, and it records these replacements in μ_0. For each $i > 0$, the chase selects a rule $r \in P$ and a substitution σ with $\sigma(\mathsf{b}(r)) \subseteq I_{i-1}$ and (i) if $\sigma(\mathsf{h}(r))$ is of the form $s \approx t$ and $s \neq t$, then one term, say s, is selected as the *representative*, and I_i and μ_i are obtained from I_{i-1} and μ_{i-1} by replacing t with s and setting $\mu_i(t) = s$; and (ii) if $\sigma(\mathsf{h}(r))$ does not contain $\approx$ and $\sigma(\mathsf{h}(r)) \notin I_{i-1}$, then $\mu_i = \mu_{i-1}$ and $I_i = I_{i-1} \cup \{\mu_i[\![\sigma(\mathsf{h}(r))]\!]\}$. The computation proceeds until no rule is applicable and then returns the final pair $\langle I_n, \mu_n \rangle$. If the representatives are always chosen as smallest in an arbitrary, but fixed well-founded order on ground terms, then the result is unique for P and B and it is called the *chase* of P on B, written $\mathsf{chase}(P, B)$. The following properties of the chase are well known (Benedikt et al. 2017).

Proposition 1. *For each program P, base instance B, $\mathsf{chase}(P, B) = \langle I, \mu \rangle$, and $P' = P \cup \mathsf{R}(P) \cup \mathsf{C}(P) \cup \mathsf{ST}$, (i) $\mu(t_1) = \mu(t_2)$ if and only if $P' \cup B \models t_1 \approx t_2$, for all ground terms t_1 and t_2, and (ii) $P' \cup B \models R(\mathbf{t})$ if and only if $R(\mu[\![\mathbf{t}]\!]) \in I$, for each ground relational atom $R(\mathbf{t})$.*

Intuitively, $\mu(t)$ is a unique *representative* of each ground term t, and the chase maintains and propagates facts only among the representative facts of $T_{P'}^{\infty}(B)$, instead of naïvely firing congruence rules. The algorithm is used in systems such as Llunatic and RDFox (Benedikt et al. 2017).

When reasoning with equality, an important question is whether programs are allowed to equate constants. We say that P and B *satisfy UNA* if $P' \cup B \models a \approx b$ implies $a = b$. Our algorithms do not require UNA to be satisfied, but certain steps can be optimized if we know that UNA is satisfied.

3 Motivation and Overview

To understand the challenges of goal-driven query answering over existential rules with $\approx$, let Σ^{ex} consist of (5)–(9).

$$A(x) \wedge R(x, y) \rightarrow \mathsf{Q}(x) \tag{5}$$

$$S(x, z) \rightarrow \exists y. R(x, y) \tag{6}$$

$$R(x, y) \wedge S(x, x') \wedge R(x', y') \rightarrow y \approx y' \tag{7}$$

$$B(x) \rightarrow \exists y. T(x, y) \wedge A(y) \tag{8}$$

$$T(x, y) \rightarrow x \approx y \tag{9}$$

Let $B^{ex} = \{B(a_1)\} \cup \{S(a_{i-1}, a_i) \mid 1 < i \leq n\}$. One can check that $\Sigma^{ex} \cup B^{ex} \models \mathsf{Q}(a_i)$ holds only for $i = 1$; however, all bottom-up techniques known to us will "fire" (6) and (7) for all a_i. In logic programming, goal-driven or top-down approaches, such as SLD resolution, start from the query and search for proofs backwards. The magic sets algorithm transforms a program so that evaluating the result bottom-up mimics top-down evaluation. These approaches are not directly applicable to existential rules, but we can apply them to the program $P' = P \cup \mathsf{R}(P) \cup \mathsf{C}(P) \cup \mathsf{ST}$, obtained by Skolemizing TGDs as $P = \mathsf{sk}(\Sigma^{ex})$ and then axiomatizing equality. This, however, is inefficient since the congruence axioms introduce many redundant proofs. In particular, Skolemizing (6) produces $R(x, f(x)) \leftarrow S(x, z)$. By rule (7), we have $P' \models f(a_{i-1}) \approx f(a_i)$ for $1 < i \leq n$ so, by the reflexivity, symmetry, and transitivity axioms for $\approx$, we have $P' \models f(a_i) \approx f(a_j)$ for $1 \leq i, j \leq n$. Hence, by the congruence axioms, we have $P' \models R(a_i, f(a_j))$. Thus, $P' \models \mathsf{Q}(a_1)$ has (at least) n proofs, where the first step uses a ground rule instance $\mathsf{Q}(a_1) \leftarrow A(a_1) \wedge R(a_1, f(a_i))$ for each $1 \leq i \leq n$. The magic sets algorithm will explore all of these proofs, which is very expensive. In contrast, our technique can answer the query by considering this rule instantiated only for $i = 1$ (see Example 5).

We present an approach that gives the benefits of top-down approaches, while radically pruning the set of considered proofs. Our approach has additional benefits. It does not require UNA, but certain steps can be optimized if $\Sigma \cup B$ satisfies UNA (e.g., if an earlier UNA check succeeded). Moreover, it preserves chase termination, and it includes an optimized magic set transformation using the symmetry of equality to greatly reducing the number of output rules.

Our technique is presented in the pipeline shown in Algorithm 1. Instead of axiomatizing equality, we first apply *singularization* (line 1), a well-known transformation that makes all relevant equalities explicit (Marnette 2009; ten Cate et al. 2009), and then we convert the result to a logic program using Skolemization (line 2). Next, we apply a relevance analysis algorithm (line 3) that identifies the rules relevant to the query. We next apply the magic sets transformation optimized for $\approx$ (line 4); the removal of irrelevant equality atoms during relevance analysis ensures that this step produces a smaller program. Finally, we remove the function symbols (line 5) and equalities (line 6) from rule bodies, obtaining a program that can be safely evaluated using the chase for logic programs (lines 7–9). We explain the components in detail in the following sections.

Algorithm 1 Compute the answers to query Q over a finite set of existential rules Σ and a base instance B

```
1: $\Sigma_1 := \mathsf{sg}(\Sigma)$
2: $P_2 := \mathsf{sk}(\Sigma_1)$
3: $P_3 := \mathsf{relevance}(P_2, B)$
4: $P_4 := \mathsf{magic}(P_3)$
5: $P_5 := \mathsf{defun}(P_4)$
6: $P_6 := \mathsf{desg}(P_5)$
7: $\langle I, \mu \rangle := \mathsf{chase}(P_6, B)$
8: for each $\mathsf{Q}(\mathbf{a}) \in I$ where $\mathbf{a}$ are constants do
9:     output each tuple of constants $\mathbf{b}$ with $\mu[\![\mathbf{b}]\!] = \mathbf{a}$
```

4 Singularization

Singularization is an alternative to congruence axioms.

Definition 1. *A* singularization *of an existential rule τ is obtained from τ by exhaustively (i) replacing each occurrence* *of a constant c in a relational body atom with a fresh variable x and adding atom $x \approx c$ to the body, and (ii) for each variable x occurring at least twice in (not necessarily distinct) relational body atoms, replacing one such occurrence with a fresh variable x' and adding atom $x' \approx x$ to the body.*

A singularization *of a set of existential rules Σ defining the query predicate Q is obtained by replacing each TGD of the form $\varphi \to \mathsf{Q}(x_1, \ldots, x_n)$ with* (10) *for $x'_1, \ldots, x'_n$ fresh variables, and then singularizing all existential rules.*

$$\varphi \wedge \bigwedge_{i=1}^{n} x_i \approx x'_i \to \mathsf{Q}(x'_1, \ldots, x'_n) \tag{10}$$

Example 1. *On Σ^{ex}, singularization leaves* (6), (8), *and* (9) *intact since their bodies do not contain repeated variables. Rules* (5) *and* (7) *are singularized as* (11) *and* (12).

$$A(x'') \wedge x \approx x'' \wedge R(x, y) \wedge x \approx x' \to \mathsf{Q}(x') \tag{11}$$

$$\begin{aligned} &R(x, y) \wedge x \approx x'' \wedge \\ &\quad S(x'', x') \wedge x' \approx x''' \wedge R(x''', y') \to y \approx y' \end{aligned} \tag{12}$$

The result of singularization is not unique: (5) could also produce $A(x) \wedge x \approx x'' \wedge R(x'', y) \wedge x \approx x' \to \mathsf{Q}(x')$. In our approach, we let $\mathsf{sg}(\Sigma)$ be any singularization of Σ.

Singularization highlights the relevant equalities originating from joins, which in turn preserves all query answers without relying on congruence axioms: for each base instance B and tuple $\mathbf{a}$ of constants, we have $\Sigma \cup B \models_{\approx} \mathsf{Q}(\mathbf{a})$ if and only if $\mathsf{sg}(\Sigma) \cup \mathsf{R}(\Sigma) \cup \mathsf{ST} \cup B \models \mathsf{Q}(\mathbf{a})$. Singularization still relies on reflexivity axioms. As an important optimization, we prove that these do not need to be analyzed in the remaining steps of our pipeline, which ensures that our pipeline produces smaller, more efficient programs. To achieve this, we show that our transformations produce rules satisfying the following condition.

Definition 2. *A rule r is* $\approx$-safe *if, for each equality atom $A \in \mathsf{b}(r)$, the atom is of the form $x \approx y$ or $x \approx s$ for s a ground term, and $\mathsf{vars}(A) \cap \mathbf{t}_i \neq \emptyset$ for some relational atom $R_i(\mathbf{t}_i) \in \mathsf{b}(r)$. A program is* $\approx$-safe *all its rules are $\approx$-safe.*

Intuitively, $\approx$-safety ensures that each fact $t \approx t$ matching a body atom of a rule r can be derived from another relational body atom of r. Thus, we do not need to pass the reflexivity axiom as input to the steps of our pipeline in order to determine which of these are pertinent to Q. Instead, the pertinent reflexivity axioms are determined directly by the predicates occurring in the result of each pipeline step.

We apply Skolemization after singularization to eliminate existential quantifiers.

Example 2. *In our running example, only* (6) *and* (8) *contain existential quantifiers, so they are replaced by* (14), *and* (16) *and* (17)*; all other rules are reinterpreted as logic programming rules. Program P_2 contains rules* (13)–(18).

$$\mathsf{Q}(x') \leftarrow A(x'') \wedge x \approx x'' \wedge R(x, y) \wedge x \approx x' \tag{13}$$

$$R(x, f(x)) \leftarrow S(x, z) \tag{14}$$

$$\begin{aligned} y \approx y' \leftarrow R(x, y) \wedge x \approx x'' \wedge S(x'', x') \wedge \\ x' \approx x''' \wedge R(x''', y') \end{aligned} \tag{15}$$

$$T(x, g(x)) \leftarrow B(x) \tag{16}$$

$$A(g(x)) \leftarrow B(x) \tag{17}$$

$$x \approx y \leftarrow T(x, y) \tag{18}$$

The answers to Q on $\Sigma \cup B$ and $P_2 \cup \mathsf{R}(P_2) \cup \mathsf{ST} \cup B$ coincide on each base instance B, but the absence of congruence axioms considerably reduces the number of proofs: EGD (7) *still ensures $P_2 \models f(a_i) \approx f(a_j)$ for $1 \leq i, j \leq n$, but $P_2 \models R(a_i, f(a_j))$ holds only for $i = j$. Thus, the only remaining proof of $P_2 \models \mathsf{Q}(a_1)$ is via the ground instance $\mathsf{Q}(a_1) \leftarrow C(a_1) \wedge R(a_1, f(a_1))$, which benefits all goal-driven techniques, including magic sets. Also, none of the facts derived by EGD* (7) *contribute to a proof of $\mathsf{Q}(a_1)$, so singularization makes EGD redundant; in Section 5 we present way to detect and eliminate such rules.*

5 Relevance Analysis

The next step of the pipeline eliminates rules all of whose consequences are irrelevant to Q. The idea is to homomorphically embed B into a much smaller instance B' called an *abstraction* of B. If B' is sufficiently small, we can analyze ways to derive answers to Q on B'; since homomorphism composition is a homomorphism, this will uncover all ways to derive an answer to Q on the original base instance B.

Definition 3. *A base instance B' is an* abstraction *of a base instance B w.r.t. a program P if there exists a homomorphism η from B to B' preserving the constants in P—that is, η maps constants to constants such that $\eta(B) \subseteq B'$ and $\eta(c) = c$ for each constant c occurring in P.*

To abstract B into B', we can use the *critical instance* for B: for C the set of constants of P and $*$ a fresh constant, we let B' contain $R(\mathbf{a})$ for each n-ary predicate R occurring in B and each $\mathbf{a} \in (C \cup \{*\})^n$. We can further refine the abstraction if predicates are *sorted* (i.e., each predicate position is associated with a sort such as strings or integers): we introduce a distinct fresh constant $*_i$ per sort, and we form B' as above while honoring the sorting requirements.

Algorithm 2 takes an $\approx$-safe program P and a base instance B, and it returns the rules relevant to answering Q on B. It selects an abstraction B' of B w.r.t. P (line 1), computes the consequences I of P on B' (line 2), and identifies the rules of P contributing to the answers of Q on B' by a form of backward chaining. It initializes the "ToDo" set $\mathcal{T}$ to all homomorphic images of the answers to Q on B' (line 3) and then iteratively explores $\mathcal{T}$ (lines 5–12). In each iteration, it extracts a fact F from $\mathcal{T}$ (line 6) and then identifies each rule r and substitution ν matching the head of r to F and the body of r in I (line 7). Such ν captures ways of deriving a fact represented by F from B via r, so r is added to the set $\mathcal{R}$ of relevant rules if such μ exists (line 8). Finally, the matched body atoms must be derivable as well, so they are all added to $\mathcal{T}$ (line 11). The "done" set $\mathcal{D}$ ensures that each fact is added to $\mathcal{T}$ just once, which ensures termination.

We can optimize the algorithm if $P \cup B$ is known to satisfy UNA (e.g., if an earlier UNA check was conducted). If ν matches an atom $x \approx t \in \mathsf{b}(r)$ as $c \approx c$, the corresponding derivation from P and B necessarily matches $x \approx t$ to $d \approx d$ for some constant d; due to $\approx$-safety, we can derive $d \approx d$ using reflexivity axioms, so we do not need to examine other proofs for $d \approx d$ (line 10). Moreover, if all matches

Algorithm 2 $\mathsf{relevance}(P, B)$

```
1: choose an abstraction $B'$ of $B$ w.r.t. $P$
2: $I := T^{\infty}_{P'}(B')$ for $P' = P \cup \mathsf{R}(P) \cup \mathsf{ST}$
3: $\mathcal{D} := \mathcal{T} := \{\mathsf{Q}(\mathbf{a}) \in I \mid \mathbf{a}$ is a tuple of constants$\}$
4: $\mathcal{R} := \emptyset$ and $\mathcal{B} := \emptyset$
5: while $\mathcal{T} \neq \emptyset$ do
6:     choose and remove some fact $F$ from $\mathcal{T}$
7:     for each $r \in P \cup \mathsf{ST}$ and substitution $\nu$ such that
           $\nu(\mathsf{h}(r)) = F$ and $\nu(\mathsf{b}(r)) \subseteq I$ do
8:         if $r \notin \mathcal{R} \cup \mathsf{ST}$ then add $r$ to $\mathcal{R}$
9:         for each $G_i \in \nu(\mathsf{b}(r))$ do
10:            if $G_i$ is not of the form $c \approx c$ with $c$ a constant, or
                   $P \cup B$ is not known to satisfy UNA then
11:                if $G_i \notin \mathcal{D}$ then add $G_i$ to $\mathcal{T}$ and $\mathcal{D}$
12:                if $G_i$ is an equality then add $\langle r, i\rangle$ to $\mathcal{B}$
13: if $P$ and $B$ and known to satisfy UNA then
14:     for each $r \in \mathcal{R}$ and each $i$-th atom of $\mathsf{b}(r)$ of the form
           $x \approx t$ with $t$ a term and $\langle r, i\rangle \notin \mathcal{B}$ do
15:         remove $x \approx t$ from $r$ and replace $x$ with $t$ in $r$
16: return $\mathcal{R}$
```

of $x \approx t$ are of such a form, then we can replace x with t in r and inform the subsequent magic sets transformation step that no equalities are relevant to this atom. To this end, Algorithm 2 maintains a set $\mathcal{B}$ of "blocked" body equality atoms that records all body equality atoms that can be matched to an equality not of the form $d \approx d$ (line 12). After considering all possibilities for deriving certain answers, body equality atoms that have not been blocked are removed (lines 13–15).

The computation of I in line 2 may not terminate in general, but we shall apply Algorithm 2 in our pipeline only in cases where termination is guaranteed. Theorem 1 shows that, in that case, the query answers remain preserved.

Theorem 1. *For each $\approx$-safe program P defining the query predicate Q and base instance B where line 2 of Algorithm 2 terminates, program $\mathcal{R} = \mathsf{relevance}(P, B)$ is $\approx$-safe, and, for each tuple $\mathbf{a}$ of constants, $P \cup \mathsf{R}(P) \cup \mathsf{ST} \cup B \models \mathsf{Q}(\mathbf{a})$ if and only if $\mathcal{R} \cup \mathsf{R}(\mathcal{R}) \cup \mathsf{ST} \cup B \models \mathsf{Q}(\mathbf{a})$.*

If computing $T^{\infty}_{P'}(B')$ in line 1 is difficult (as is the case in some of our experiments), we can replace each term $f(\mathbf{x})$ in P with a fresh constant c_f unique for f. This does not affect the algorithm's correctness, since $T^{\infty}_{P'}(B)$ can still be homomorphically embedded into $T^{\infty}_{P'}(B')$. Moreover, the resulting program then does not contain function symbols, and so the computation in line 2 necessarily terminates.

Example 3. *In our running example, UNA holds on Σ^{ex} and B^{ex}, and we shall assume that this is known in advance. Moreover, we shall take B' to be the critical instance, containing facts $B(*)$ and $S(*, *)$. Computing the least fixpoint in line 2 of Algorithm 2 produces the following instance I.*

$$\begin{array}{cccc} B(*) & S(*,*) & R(*, f(*)) & f(*) \approx f(*) \\ T(*, g(*)) & A(g(*)) & * \approx * & * \approx g(*) \\ g(*) \approx * & g(*) \approx g(*) & \mathsf{Q}(*) & \mathsf{Q}(g(*)) \end{array}$$

Algorithm 2 starts by considering $\mathsf{Q}(*)$. Matching the fact to the head of (13) *and evaluating the body in I produces the ground rule instance*

$$\mathsf{Q}(*) \leftarrow A(g(*)) \wedge * \approx g(*) \wedge R(*, f(*)) \wedge * \approx *;$$

thus, rule (13) *is identified as relevant. UNA is known to hold, so atom $* \approx *$ is not considered any further, but the algorithm must consider the remaining body atoms. Matching $g(*) \approx f(*)$ to the head of* (18) *and evaluating the body in I produces the ground rule instance $* \approx g(*) \leftarrow T(*, g(*))$, so rule* (18) *is identified as relevant; moreover, matching $T(*, g(*))$ to the head of* (16) *produces the ground rule instance $T(*, g(*)) \leftarrow B(*)$, so rule* (16) *is identified as relevant as well. In contrast, matching $* \approx g(*)$ to the head of* (15) *produces query*

$$R(x, *) \wedge \approx x'' \wedge S(x'', x') \wedge x' \approx x''' \wedge R(x''', g(*)),$$

which has no matches in I; thus, rule (15) *is not added to $\mathcal{R}$. Next, matching $R(*, f(*))$ to the head of* (14) *and evaluating the body in I produces the ground rule instance $R(*, f(*)) \leftarrow S(*, *)$; thus, rule* (14) *is identified as relevant. Finally, $B(*)$ and $S(*, *)$ do not match any rule head, so the algorithm terminates. Thus, the algorithm returns all rules apart from* (15). *Moreover, atom $x \approx x''$ in* (13) *is matched to $* \approx f(*)$ so it cannot be removed—that is, this equality is relevant. In contrast, atom $x \approx x'$ in* (13) *is matched* only *to $* \approx *$; since we know that UNA holds, this equality is irrelevant and it is removed in lines 13–15 of Algorithm 2. Consequently, the algorithm returns program P_3 consisting of rules* (19)–(23).

$$\mathsf{Q}(x) \leftarrow A(x'') \wedge x \approx x'' \wedge R(x, y) \qquad (19)$$
$$R(x, f(x)) \leftarrow S(x, z) \qquad (20)$$
$$T(x, g(x)) \leftarrow B(x) \qquad (21)$$
$$A(g(x)) \leftarrow B(x) \qquad (22)$$
$$x \approx y \leftarrow T(x, y) \qquad (23)$$

6 Magic Sets for Existential Rules with $\approx$

We now present our variant of the magic sets transformation. Our technique also mimics top-down evaluation, but with a specialized treatment of equality that takes the symmetry of $\approx$ into account and further prunes the set of proofs for facts of the form $t \approx t$. In particular, singularization critically relies on reflexivity axioms, and passing these to the magic sets algorithm would significantly blow up the resulting rule set. The $\approx$-safety of the rules implies that our magic transformation does not need to be applied to the reflexivity rules, which results in a much more efficient magic program.

We follow Beeri and Ramakrishnan (1991) in defining adornments and magic predicates for predicates other than $\approx$, but, as we discuss shortly, we optimize these notions for $\approx$. Intuitively, an adornment identifies which arguments of an atom will be bound by sideways information passing, and a magic predicate will "collect" the passed arguments.

Definition 4. *An* adornment *for an n-ary predicate R other than $\approx$ is a string α of length n over alphabet* b *("bound") and* f *("free"), and m^{α}_{R} is a fresh* magic predicate *unique for R and α with arity equal to the number of* b*-symbols in α. An* adornment *for $\approx$ has the form* bb, bf, *or* fb, *and $m^{\mathsf{bb}}_{\approx}$ and $m^{\mathsf{b}\wr\mathsf{f}}_{\approx}$ are fresh* magic *predicates for $\approx$ of arity two and one, respectively. For α an adornment of length n and $\mathbf{t}$ an n-tuple of terms, $\mathbf{t}^{\alpha}$ contains in the same relative order each $t_i \in \mathbf{t}$ for which the i-th element of α is* b.

Definition 4 takes into account that, if one argument of an equality atom is bound, the other argument will also be bound due to the symmetry of $\approx$. Thus, predicate $m_{\approx}^{\mathsf{b}\wr\mathsf{f}}$ is used for both bf and fb, where notation $\mathsf{b}\wr\mathsf{f}$ stresses that the positions of b and f are interchangeable. Moreover, at least one argument of an equality atom must be bound, so $\approx$ cannot be adorned by ff. Definition 5 introduces a sideways information passing strategy (SIPS), which determines how information is propagated through the rule bodies. Function $\mathsf{reorder}$ reorders the rule bodies to maximize information passing, and function adorn decides which arguments of an atom should be bound given a set of available bindings.

Definition 5. *A* sideways information passing strategy *consists of the following two functions.*

For φ a conjunction of atoms and T a set of terms, $\mathsf{reorder}(\varphi, T)$ returns an ordering $\langle R_1(\mathbf{t}_1), \dots, R_n(\mathbf{t}_n)\rangle$ of the conjuncts of φ such that, for each equality atom $R_i(\mathbf{t}_i)$ of the form $x \approx y$ or $x \approx s$ where s is ground, $z \in \mathsf{vars}(\mathbf{t}_i)$ exists such that $z \in T$ or $z \in \mathbf{t}_j$ for some $j < i$ with $R_j \neq \approx$.

For $R(t_1, \dots, t_k)$ an atom and V a set of variables, $\mathsf{adorn}(R(t_1, \dots, t_k), V)$ returns an adornment α for R such that $\mathsf{vars}(t_j) \subseteq V$ if the j-th element of α is b.

Algorithm 3 implements the magic sets transformation optimized for $\approx$. It initializes the "ToDo" set $\mathcal{T}$ with the magic predicate for the query (line 1) and processes $\mathcal{T}$ iteratively. For each magic predicate m_R^α in $\mathcal{T}$ (line 3), it identifies each rule r that can derive R (line 4). Adornment $\alpha = \mathsf{b}\wr\mathsf{f}$ is processed as both bf and fb (lines 6–7), and each other α is processed as is (line 9). In all cases, the algorithm produces the *modified rule* by restricting the body of r by the magic predicate corresponding to the head predicate (line 12), and then it reorders (line 13) and processes (lines 14–19) the body of r. For each body atom $R_i(\mathbf{t}_i)$ with R_i occurring in the head of P, the algorithm uses the SIPS to determine an adornment γ identifying the bound arguments (line 15), and it generates the *magic rule* that populates $m_{\approx}^{\mathsf{b}\wr\mathsf{f}}$ or $m_{R_i}^{\gamma}$ with the bindings for R_i (line 18). The algorithm takes into account that $m_{\approx}^{\mathsf{b}\wr\mathsf{f}}$ captures both bf and fb (line 16–17). The magic rule would not be $\approx$-safe if $R_i(\mathbf{t}_i)$ were an equality atom with no arguments bound, which, at the end of our pipeline, could produce a rule with variables occurring the head but not the body. Thus, Definition 4 does not introduce the ff adornment for $\approx$, and Definition 5 requires at least one argument of each equality atom to be bound. Finally, the magic predicate must also be processed (line 19), and the "done" set $\mathcal{D}$ ensures that this happens just once.

Theorem 2 shows that Algorithm 3 preserves $\approx$-safety and query answers, and Theorem 3 shows that it also preserves chase termination. The latter does not hold for all programs with function symbols; for example, transforming $A(x) \leftarrow A(f(x))$ can produce the nonterminating rule $m_A^f(f(x)) \leftarrow m_A^f(x)$. However, in Algorithm 1, the magic sets are applied in line 4 only to programs P_3 that do not contain function symbols in the body, which suffices to show that the heads of the magic rules are function-free and cannot derive terms of unbounded depth, and therefore the transformation does not affect termination.

Algorithm 3 $\mathsf{magic}(P)$

```
 1: $\mathcal{D} := \mathcal{T} := \{m_{\mathsf{Q}}^\alpha\}$ and $\mathcal{R} := \{m_{\mathsf{Q}}^\alpha \leftarrow\}$,   $\alpha = \mathsf{f}\cdots\mathsf{f}$
 2: while $\mathcal{T} \neq \emptyset$ do
 3:     choose and remove some $m_R^\alpha$ from $\mathcal{T}$
 4:     for each $r \in P \cup \mathsf{ST}$ such that $\mathsf{h}(r) = R(\mathbf{t})$ do
 5:         if $R = \approx$ and $\alpha = \mathsf{b}\wr\mathsf{f}$ then
 6:             $\mathsf{process}(r, \alpha, \mathsf{bf})$
 7:             $\mathsf{process}(r, \alpha, \mathsf{fb})$
 8:         else
 9:             $\mathsf{process}(r, \alpha, \alpha)$
10: return $\mathcal{R}$

11: procedure $\mathsf{process}(r, \alpha, \beta)$ where $\mathsf{h}(r) = R(\mathbf{t})$
12:     add $\mathsf{h}(r) \leftarrow m_R^\alpha(\mathbf{t}^\beta) \wedge \mathsf{b}(r)$ to $\mathcal{R}$
13:     $\langle R_1(\mathbf{t}_1), \dots, R_n(\mathbf{t}_n)\rangle := \mathsf{reorder}(\mathsf{b}(r), \mathbf{t}^\beta)$
14:     for $1 \leq i \leq n$ s.t. $R_i$ is $\approx$ or it occurs in $P$ in a head do
15:         $\gamma := \mathsf{adorn}(R_i(\mathbf{t}_i), \mathsf{vars}(\mathbf{t}^\beta) \cup \bigcup_{j=1}^{i-1} \mathsf{vars}(\mathbf{t}_j))$
16:         if $R_i = \approx$ and $\gamma \in \{\mathsf{bf}, \mathsf{fb}\}$ then $S := m_{\approx}^{\mathsf{b}\wr\mathsf{f}}$
17:         else $S := m_{R_i}^\gamma$
18:         add $S(\mathbf{t}_i^\gamma) \leftarrow m_R^\alpha(\mathbf{t}^\beta) \wedge \bigwedge_{j=1}^{i-1} R_j(\mathbf{t}_j)$ to $\mathcal{R}$
19:         if $S \notin \mathcal{D}$ then add $S$ to $\mathcal{T}$ and $\mathcal{D}$
```

Note: please remember that $t_1 \approx t_2$ can also be written as $\approx(t_1, t_2)$, so $R(\mathbf{t})$ and $R_i(\mathbf{t}_i)$ can be equality atoms.

Theorem 2. *For each $\approx$-safe program P defining the query predicate Q, program $\mathcal{R} = \mathsf{magic}(P)$ is $\approx$-safe; moreover, for each base instance B and each tuple $\mathbf{t}$ of ground terms,*

$$P \cup \mathsf{R}(P) \cup \mathsf{ST} \cup B \models \mathsf{Q}(\mathbf{t}) \quad \textit{iff} \quad \mathcal{R} \cup \mathsf{R}(\mathcal{R}) \cup \mathsf{ST} \cup B \models \mathsf{Q}(\mathbf{t}).$$

Theorem 3. *Let P be a program where the body of each rule is function-free, and let $P_1 = P \cup \mathsf{R}(P) \cup \mathsf{ST}$ and $P_2 = \mathcal{R} \cup \mathsf{R}(\mathcal{R}) \cup \mathsf{ST}$ for $\mathcal{R} = \mathsf{magic}(P)$. For each base instance B, if $T_{P_1}^\infty(B)$ is finite, then $T_{P_2}^\infty(B)$ is finite as well.*

Example 4. *Applying Algorithm 3 to P_3 produces program P_4 consisting of rules* (24)–(36). *Horizontal lines separate the rules produced in each invocation of* $\mathsf{process}$. *Please note that* (23) *produces* (30) *and* (31) *when* $\mathsf{b}\wr\mathsf{f}$ *is interpreted as* bf, *and* (30) *and* (31) *when* $\mathsf{b}\wr\mathsf{f}$ *is interpreted as* bf.

$$m_{\mathsf{Q}}^{\mathsf{f}} \leftarrow \tag{24}$$

$$\mathsf{Q}(x) \leftarrow m_{\mathsf{Q}}^{\mathsf{f}} \wedge A(x'') \wedge x \approx x'' \wedge R(x, y) \tag{25}$$

$$m_A^{\mathsf{f}} \leftarrow m_{\mathsf{Q}}^{\mathsf{f}} \tag{26}$$

$$m_{\approx}^{\mathsf{b}\wr\mathsf{f}}(x'') \leftarrow m_{\mathsf{Q}}^{\mathsf{f}} \wedge A(x'') \tag{27}$$

$$m_R^{\mathsf{bf}}(x) \leftarrow m_{\mathsf{Q}}^{\mathsf{f}} \wedge A(x'') \wedge x \approx x'' \tag{28}$$

$$A(g(x)) \leftarrow m_A^{\mathsf{f}} \wedge B(x) \tag{29}$$

$$x \approx y \leftarrow m_{\approx}^{\mathsf{b}\wr\mathsf{f}}(x) \wedge T(x, y) \tag{30}$$

$$m_T^{\mathsf{bf}}(x) \leftarrow m_{\approx}^{\mathsf{b}\wr\mathsf{f}}(x) \tag{31}$$

$$x \approx y \leftarrow m_{\approx}^{\mathsf{b}\wr\mathsf{f}}(y) \wedge T(x, y) \tag{32}$$

$$m_T^{\mathsf{fb}}(y) \leftarrow m_{\approx}^{\mathsf{b}\wr\mathsf{f}}(y) \tag{33}$$

$$T(x, g(x)) \leftarrow m_T^{\mathsf{bf}}(x) \wedge B(x) \tag{34}$$

$$T(x, g(x)) \leftarrow m_T^{\mathsf{fb}}(g(x)) \wedge B(x) \tag{35}$$

$$R(x, f(x)) \leftarrow m_R^{\mathsf{bf}}(x) \wedge S(x, z) \tag{36}$$

7 Final Transformations

The final steps ensure that the resulting program can be evaluated efficiently using the chase for logic programs, which, as explained in Section 2, can handle only programs with no constants, function symbols, and $\approx$ in the rule bodies. The magic sets transformation can introduce body atoms with function symbols, so Definition 6 removes these by introducing fresh predicates. Proposition 2 shows the query answers remain preserved since the fresh predicates can always be interpreted to reflect the structure of the ground functional terms encountered during the chase.

Definition 6. *Program* $\mathsf{defun}(P)$ *is obtained from a program* P *by exhaustively applying the following steps.*

1. *In the body of each rule, replace each occurrence of a constant* c *with a fresh variable* z_c *unique for* c*, add atom* $\mathsf{F}_c(z_t)$ *to the body, and add the rule* $\mathsf{F}_c(c) \leftarrow$*.*
2. *In the body of each rule, replace each occurrence of a term* t *of the form* $f(\mathbf{s})$ *with a fresh variable* z_t *unique for* t*, and add atom* $\mathsf{F}_f(\mathbf{s}, z_t)$ *to the body.*
3. *For each rule* r *and each term of the form* $f(\mathbf{s})$ *occurring in* $\mathsf{h}(r)$ *with* f *a function symbol considered in the second step, add the rule* $\mathsf{F}_f(\mathbf{s}, f(\mathbf{s})) \leftarrow \mathsf{b}(r)$*.*

Proposition 2. *For each program* P *and* $P' = \mathsf{defun}(P)$*, base instance* B*, predicate* R *not of the form* F_f*, and tuple* $\mathbf{t}$ *of ground terms,* $P \cup \mathsf{R}(P) \cup \mathsf{ST} \cup B \models R(\mathbf{t})$ *if and only if* $P' \cup \mathsf{R}(P') \cup \mathsf{ST} \cup B \models R(\mathbf{t})$*.*

Definition 7 reverses the effects of singularization and removes all body equality atoms. As a side-effect, this reduces the number of rule variables, which simplifies rule matching.

Definition 7. *The* desingularization *of a rule is obtained by repeatedly removing each body atom of the form* $x \approx t$ *while replacing* x *with* t *everywhere in the rule. For* P *a program,* $\mathsf{desg}(P)$ *contains a desingularization of each rule of* P*.*

We evaluate the final program using the chase for logic programs, which captures the effects of congruence axioms. Note, however, that program P_5 from Algorithm 1 contains fresh predicates introduced by the magic sets transformation and the elimination of function symbols, to which the chase will (implicitly) apply congruence axioms as well. Theorem 4 shows that this preserves the query answers, and its proof is not trivial: adding congruence axioms to a program produces new consequences, so the proof depends on the fact program P_5 was obtained as shown in Algorithm 1.

Theorem 4. *For each finite set of existential rules* Σ *defining the query predicate* Q*, each base instance* B*, each tuple of constants* $\mathbf{a}$*, and program* P_6 *obtained from* Σ *and* B *by applying Algorithm 1,* $\Sigma \cup B \models_\approx \mathsf{Q}(\mathbf{a})$ *if and only if* $P_6 \cup \mathsf{R}(P_6) \cup \mathsf{C}(P_6) \cup B \models \mathsf{Q}(\mathbf{a})$*.*

Theorem 5 shows that our entire pipeline is correct. Note that, if the Skolem chase of $\mathsf{sg}(\Sigma)$ terminates on every base instance, then line 2 of Algorithm 2 necessarily terminates.

Theorem 5. *For each finite set of existential rules* Σ *defining the query predicate* Q *such that the chase of* $\mathsf{sg}(\Sigma)$ *terminates on all base instances, and for each base instance* B*, Algorithm 1 outputs precisely all answers to* Q *on* $\Sigma \cup B$ *and then terminates.*

Example 5. *In our running example, program* P_6 *contains rules* (37)–(52)*, where* (34) *produces* (48) *and* (49)*, and* (35) *produces* (50) *and* (51)*.*

$$m^{\mathsf{f}}_{\mathsf{Q}} \leftarrow \tag{37}$$

$$\mathsf{Q}(x) \leftarrow m^{\mathsf{f}}_{\mathsf{Q}} \wedge A(x) \wedge R(x,y) \tag{38}$$

$$m^{\mathsf{f}}_{A} \leftarrow m^{\mathsf{f}}_{\mathsf{Q}} \tag{39}$$

$$m^{\mathsf{b}\wr\mathsf{f}}_{\approx}(x'') \leftarrow m^{\mathsf{f}}_{\mathsf{Q}} \wedge A(x'') \tag{40}$$

$$m^{\mathsf{bf}}_{R}(x'') \leftarrow m^{\mathsf{f}}_{\mathsf{Q}} \wedge A(x'') \tag{41}$$

$$A(g(x)) \leftarrow m^{\mathsf{f}}_{A} \wedge B(x) \tag{42}$$

$$\mathsf{F}_g(x, g(x)) \leftarrow m^{\mathsf{f}}_{A} \wedge B(x) \tag{43}$$

$$x \approx y \leftarrow m^{\mathsf{b}\wr\mathsf{f}}_{\approx}(x) \wedge T(x,y) \tag{44}$$

$$m^{\mathsf{bf}}_{T}(x) \leftarrow m^{\mathsf{b}\wr\mathsf{f}}_{\approx}(x) \tag{45}$$

$$x \approx y \leftarrow m^{\mathsf{b}\wr\mathsf{f}}_{\approx}(y) \wedge T(x,y) \tag{46}$$

$$m^{\mathsf{fb}}_{T}(y) \leftarrow m^{\mathsf{b}\wr\mathsf{f}}_{\approx}(y) \tag{47}$$

$$T(x, g(x)) \leftarrow m^{\mathsf{bf}}_{T}(x) \wedge B(x) \tag{48}$$

$$\mathsf{F}_g(x, g(x)) \leftarrow m^{\mathsf{bf}}_{T}(x) \wedge B(x) \tag{49}$$

$$T(x, g(x)) \leftarrow \mathsf{F}_g(x, z_{g(x)}) \wedge m^{\mathsf{fb}}_{T}(z_{g(x)}) \wedge B(x) \tag{50}$$

$$\mathsf{F}_g(x, g(x)) \leftarrow \mathsf{F}_g(x, z_{g(x)}) \wedge m^{\mathsf{fb}}_{T}(z_{g(x)}) \wedge B(x) \tag{51}$$

$$R(x, f(x)) \leftarrow m^{\mathsf{bf}}_{R}(x) \wedge S(x,z) \tag{52}$$

Program P_6 *contains no constants, function symbols, or equality atoms in the body, so we can answer* Q *on* P_6 *and* B^{ex} *using the chase for logic programs and thus avoid computing the least fixpoint for a program that explicitly axiomatizes equality. Doing so derives the following facts.*

$$\begin{array}{llll} m^{\mathsf{f}}_{\mathsf{Q}} & m^{\mathsf{f}}_{A} & A(g(a_1)) & \mathsf{F}_g(a_1, g(a_1)) \\ m^{\mathsf{b}\wr\mathsf{f}}_{\approx}(g(a_1)) & m^{\mathsf{fb}}_{T}(g(a_1)) & T(a_1, g(a_1)) & \end{array}$$

Next, rule (46) *derives* $a_1 \approx g(a_1)$*, so the chase for logic programs takes* a_1 *as the representative of* $g(a_1)$ *and replaces* $g(a_1)$ *with* a_1*, thus deriving the following facts.*

$$\begin{array}{llll} m^{\mathsf{f}}_{\mathsf{Q}} & m^{\mathsf{f}}_{A} & A(a_1) & \mathsf{F}_g(a_1, a_1) \\ m^{\mathsf{b}\wr\mathsf{f}}_{\approx}(a_1) & m^{\mathsf{fb}}_{T}(a_1) & T(a_1, a_1) & \end{array}$$

After this, the chase further derives the following facts.

$$m^{\mathsf{bf}}_{T}(a_1) \qquad m^{\mathsf{bf}}_{R}(a_1) \qquad R(x, f(a_1)) \qquad \mathsf{Q}(a_1)$$

Note that no facts involving a_i *with* $i \geq 2$ *are derived, as these are irrelevant to answering* Q*. Thus, evaluating* P_6 *on* B^{ex} *produces far fewer facts than applying the chase for logic programs to* $\mathsf{sk}(\Sigma^{ex})$*.*

8 Empirical Evaluation

We evaluated our technique using ChaseBench (Benedikt et al. 2017), a recent benchmark offering a mix of scenarios that simulate data exchange and ontology reasoning applications. We selected the scenarios summarized in Table 1, each comprising a set of existential rules, a base instance, and several queries. LUBM-100 and LUBM-1K are derived from the well-known Semantic Web LUBM (Guo, Pan, and Heflin 2011) benchmark; Deep300 is a “stress test” scenario; Doctors-1M simulates data exchange between

	TGDs	EGDs	Facts	Queries	
				free	const.
LUBM-100	136	0	12 M	4	10
LUBM-1K	136	0	120 M	4	10
DEEP300	1,300	0	1 k	20	0
DOCTORS-1M	5	4	1 M	7	11
STB-128	199	93	1 M	24	26
ONT-256	529	348	2 M	34	8

Note: "const." and "free" are the numbers of queries with and without constants, respectively.

Table 1: Summary of the test scenarios

medical databases; and STB-128 and ONT-256 were produced using the IBENCH and TOXGENE rule and instance generators. The first three scenarios contain only TGDs, and the remaining ones contain EGDs as well. All rules are weakly acyclic, so the chase always terminates. Finally, UNA was known to hold in all cases (Benedikt et al. 2017).

To compute the chase of the final program (line 7 of Algorithm 1), we used the RAM-based RDFox system written in C++.[1] We implemented our technique in Java on top of the CHASEBENCH (Benedikt et al. 2017) library. We used just one thread while computing the chase. Our system and the test data are available online.[2]

No existing goal-driven query answering techniques can handle these scenarios, as explained in the introduction. Thus, we primarily compared the performance of computing the chase with no optimizations (MAT), with the pipeline that uses just the magic sets and omits the relevance analysis (MAG), with the pipeline that uses just the relevance analysis and omits the magic sets (REL), and the entire pipeline with both relevance analysis and magic sets (REL+MAG). The MAT variant thus provides us with a baseline, and the remaining tests allow us to identify the relative contribution of various steps of our pipeline. Note that, in the presence of EGDs, we always used singularization and the other relevant steps of our pipeline, rather than the congruence axioms. On TGDs only, our algorithm becomes equivalent to the classical algorithm of Beeri and Ramakrishnan (1991).

In each test run, we computed the program P_6 from Algorithm 1 (skipping the relevance analysis and/or magic sets, as required for the test type), computed $\mathsf{chase}(P_6, B)$, and output the certain answers of Q as shown in line 9 of Algorithm 1. We recorded the wall-clock time of each run (without the loading times) and the number of facts derived by the chase; the latter provides an implementation-independent measure of the work needed to answer a query. In line 1 of Algorithm 2, we abstracted the base instance using the typed critical instance; however, computing the least fixpoint of such an abstraction was infeasible on DEEP300 so, in this case only, we used the optimization from Section 5.

Figure 1 summarizes the query times and the numbers of derived facts for our 158 test queries. The whiskers of each box plot show the minimum and maximum values, the box shows the lower quartile, the median, and the upper quartile, and the diamond shows the average. The distributions of these values are shown in more detail in the appendix of the extended version (Benedikt, Motik, and Tsamoura 2017). Table 2 shows the times for computing the least fixpoint of the abstraction in line 2 of Algorithm 2, which are insignificant in all cases apart from DEEP300. On REL and REL+MAG, one query of DEEP300 and nine queries of STB-128 could not be processed by the relevance analysis for reasons we discuss shortly. Moreover, all queries of LUBM-1K and DEEP300 on MAT, three queries of LUBM-1K and one of DEEP300 on MAG, all queries of LUBM-1K and one of DEEP300 on REL, and three queries of LUBM-1K on REL+MAG could not be processed due to memory exhaustion while computing the chase.

Figure 1 clearly shows that our technique is generally very effective and can mean the difference between success and failure: the chase for LUBM-1K and DEEP300 could not be computed on our test machine, whereas REL+MAG can answer 11 out of 14 queries on LUBM-1K in at most 45 s, and 19 out of 20 queries on DEEP300 in at most 18 s. The upper quartile of query times for REL+MAG is at least an order of magnitude below the times for MAT in all cases apart from DOCTORS-1M, where this holds for the median. Overall, REL+MAG achieves the best performance.

In addition, relevance analysis alone can lead to significant improvements: the query times for REL and REL+MAG are almost identical on DEEP300 and ONT-256, suggesting that the improvements are due to relevance analysis, rather than magic sets. On some queries of STB-128 and ONT-256, the relevance analysis eliminates all rules and thus proves that queries have no answers. The benefits of relevance analysis are marginal only on LUBM, mainly because its TGDs contain few existential quantifiers.

However, relevance analysis also has its pitfalls: we could not run it on one query of DEEP300 with eight body atoms, and on nine queries of STB-128 containing between 11 and 19 output variables that, after singularization, have between 19 and 22 body atoms. Atoms of these queries match to many facts in the least fixpoint of the abstraction, so query evaluation explodes either in line 2 or line 7 of Algorithm 2. One additional query of DEEP300 exhibited similar issues, which we addressed by (manually) tree-decomposing the query and thus reducing the number of matches.

To investigate the cases in which magic sets are particularly beneficial, Table 3 shows the minimum, maximum, and median of the query times and the numbers of derived facts for queries without and with constants. The maximum numbers of derived facts are particularly telling: without constants, REL+MAG derives more facts compared to REL; and with constants, the numbers for REL+MAG are several orders of magnitude smaller compared to REL. Programs P_3 in line 3 of Algorithm 1 are the same in both cases so this improvement is clearly due to magic sets. In contrast, the impact of constants is insignificant for REL, highlighting the different strengths of relevance analysis and magic sets.

Unfortunately, magic sets are not "free". For example, MAT and REL are faster than both MAG and REL+MAG on seven constant-free queries of DOCTORS-1M, and they outperform MAG on 17 queries of STB-128 and on 39 queries

[1]http://www.cs.ox.ac.uk/isg/tools/RDFox/
[2]http://github.com/tsamoura/chaseGoal

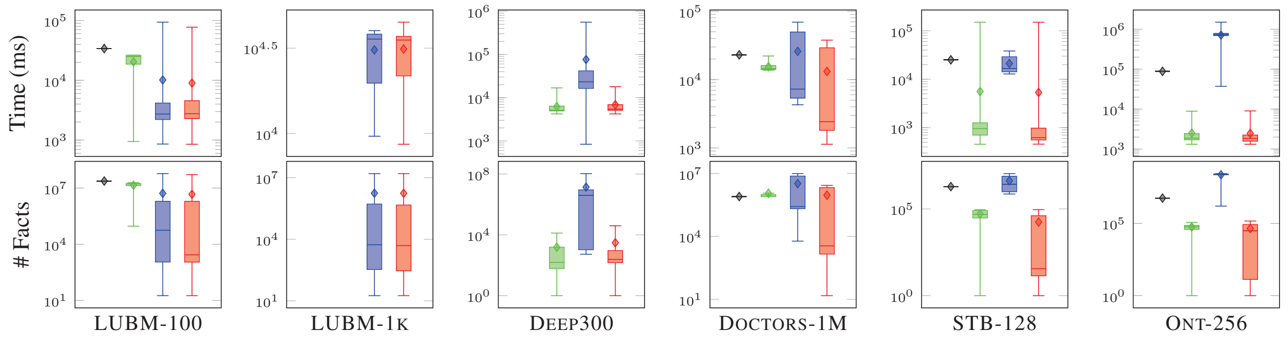

Figure 1: Times and the numbers of derived facts. Black, green, blue and red are MAT, REL, MAG and REL+MAG, respectively.

	LUBM	DEEP300	DOCTORS-1M	STB-128	ONT-256
Time	10 ms	4700 ms	10 ms	200 ms	650 ms

Table 2: Times for computing the least fixpoints of the base instance abstractions

of ONT-256. In all these cases, the magic sets transformation increases the number of rules by one or two orders of magnitude, which introduces considerable overhead during chase computation. This can be particularly significant on queries without constants: such queries tend to have more answers and thus require exploring larger proofs.

Both relevance analysis and magic sets try to identify proofs deriving a goal. Magic sets do this "at runtime": the transformed rules derive only facts that would be explored by top-down reasoning. In contrast, relevance analysis identifies rules that can participate in such proofs "offline" by checking whether all body atoms of a rule are derivable. Combining the two optimizations is particularly effective at reducing the overheads described in the previous paragraph.

9 Conclusion & Outlook

We presented a novel approach for goal-driven query answering over terminating existential rules. Our empirical results clearly show that our technique can lead to significant performance improvements and can mean the difference between success and failure to answer a query. In the future, we shall investigate the use of magic sets for query answering on nonterminating, but decidable classes of existential rules (e.g., guarded, linear, or sticky). We shall also consider adding optimizations such as tabling and subsumption.

Acknowledgments

This research was funded by the EPSRC projects DBOnto, ED^3, MaSI^3, PDQ, and the Alan Turing Institute.

References

Alviano, M.; Faber, W.; Greco, G.; and Leone, N. 2012a. Magic Sets for disjunctive Datalog programs. *Artificial Intelligence* 187:156–192.

Alviano, M.; Leone, N.; Manna, M.; Terracina, G.; and Veltri, P. 2012b. Magic-Sets for Datalog with Existential Quantifiers. In *Datalog 2.0*.

Alviano, M.; Greco, G.; and Leone, N. 2011. Dynamic Magic Sets for Programs with Monotone Recursive Aggregates. In *LPNMR*.

Baget, J.-F.; Mugnier, M.-L.; Rudolph, S.; and Thomazo, M. 2011a. Walking the complexity lines for generalized guarded existential rules. In *IJCAI*.

Baget, J.; Leclère, M.; Mugnier, M.; and Salvat, E. 2011b. On rules with existential variables: Walking the decidability line. *Artificial Intelligence* 175(9–10):1620–1654.

Baget, J.; Bienvenu, M.; Mugnier, M.; and Rocher, S. 2015a. Combining Existential Rules and Transitivity: Next Steps. In *IJCAI*.

Baget, J.; Leclère, M.; Mugnier, M.; Rocher, S.; and Sipieter, C. 2015b. Graal: A Toolkit for Query Answering with Existential Rules. In *RuleML*.

Bancilhon, F.; Maier, D.; Sagiv, Y.; and Ullman, J. D. 1986. Magic Sets and Other Strange Ways to Implement Logic Programs. In *PODS*.

Beeri, C., and Ramakrishnan, R. 1991. On the Power of Magic. *Journal of Logic Programming* 10(3&4):255–299.

Benedikt, M.; Konstantinidis, G.; Mecca, G.; Motik, B.; Papotti, P.; Santoro, D.; and Tsamoura, E. 2017. Benchmarking the Chase. In *PODS*.

Benedikt, M.; Motik, B.; and Tsamoura, E. 2017. Goal-Driven Query Answering for Existential Rules with Equality. *CoRR* abs/1711.05227.

Bonifati, A.; Ileana, I.; and Linardi, M. 2017. ChaseFUN: a Data Exchange Engine for Functional Dependencies at Scale. In *EDBT*.

Calì, A.; Gottlob, G.; and Lukasiewicz, T. 2012. A general Datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics* 14:57–83.

Calì, A.; Gottlob, G.; and Pieris, A. 2010. Query Answering under Non-guarded Rules in Datalog$^\pm$. In *RR*.

Calimeri, F.; Cozza, S.; Ianni, G.; and Leone, N. 2009.

		Query times (seconds)								
		MAG			REL			REL+MAG		
		min	max	median	min	max	median	min	max	median
LUBM-100	N	1.62	93.56	11.43	1.69	26.24	25.29	1.59	77.14	11.30
	Y	0.86	2.98	2.58	0.95	26.29	24.92	0.85	3.22	2.68
LUBM-1K	N	19.74	19.74	19.74	N/A	N/A	N/A	19.74	19.74	19.74
	Y	9.64	40.10	36.19	N/A	N/A	N/A	8.66	44.93	35.79
DOCTORS-1M	N	47.27	69.21	54.86	14.05	22.21	16.11	25.57	37.81	29.81
	Y	4.28	8.51	5.66	13.75	15.57	14.27	1.15	2.70	2.10
STB-128	N	14.65	38.35	30.07	0.45	150.57	0.74	0.46	150.21	0.75
	Y	12.87	17.92	14.35	0.75	18.47	1.10	0.51	18.18	0.59
ONT-256	N	60.36	1.49 k	745.17	1.32	8.91	1.86	1.32	9.04	1.99
	Y	37.27	703.26	690.82	1.69	2.69	1.83	1.51	2.28	1.55

		The numbers of derived facts								
		MAG			REL			REL+MAG		
		min	max	median	min	max	median	min	max	median
LUBM-100	N	1.59 M	59.23 M	5.58 M	1.59 M	18.42 M	1.74 M	1.59 M	52.41 M	5.33 M
	Y	18	51.94 k	4.70 k	92.34 k	19.52 M	17.37 M	18.00	46.40 k	4.40 k
LUBM-1K	N	15.84 M	15.84 M	15.84 M	N/A	N/A	N/A	15.84 M	15.84 M	15.84 M
	Y	18	2.77 M	4.40 k	N/A	N/A	N/A	18.00	2.77 M	4.40 k
DOCTORS-1M	N	6.65 M	9.82 M	7.44 M	953.50 k	1.74 M	1.74 M	1.90 M	2.69 M	2.69 M
	Y	5.94 k	264.30 k	263.16 k	952.50 k	953.63 k	952.50 k	15.00	4.08 k	2.96 k
STB-128	N	1.88 M	10.80 M	7.76 M	0.00	90.05 k	39.47 k	1.00	90.64 k	40.00 k
	Y	723.19 k	2.92 M	1.05 M	30.00 k	90.00 k	65.00 k	9.00	51.00	20.00
ONT-256	N	6.35 M	283.68 M	254.05 M	0.00	118.09 k	59.00 k	1.00	147.49 k	45.91 k
	Y	1.59 M	224.23 M	224.22 M	29.36 k	118.07 k	54.12 k	9.00	41.00	11.50

Table 3: Running times and the numbers of facts for queries without ("N") and with ("Y") constants

Magic Sets for the Bottom-Up Evaluation of Finitely Recursive Programs. In *LPNMR*.

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

Calvanese, D.; Cogrel, B.; Komla-Ebri, S.; Kontchakov, R.; Lanti, D.; Rezk, M.; Rodriguez-Muro, M.; and Xiao, G. 2017. Ontop: Answering SPARQL queries over relational databases. *Semantic Web* 8(3):471–487.

Deutsch, A.; Nash, A.; and Remmel, J. B. 2008. The chase revisited. In *PODS*.

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

Fagin, R.; Kolaitis, P. G.; Miller, R. J.; and Popa, L. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336(1):89–124.

Geerts, F.; Mecca, G.; Papotti, P.; and Santoro, D. 2014. That's All Folks! LLUNATIC Goes Open Source. *PVLDB* 7(13):1565–1568.

Gottlob, G.; Manna, M.; and Pieris, A. 2015. Polynomial Rewritings for Linear Existential Rules. In *IJCAI*.

Gottlob, G.; Orsi, G.; and Pieris, A. 2014. Query Rewriting and Optimization for Ontological Databases. *ACM TODS* 39(3):25:1–25:46.

Gottlob, G.; Rudolph, S.; and Simkus, M. 2014. Expressiveness of guarded existential rule languages. In *PODS*.

Grau, B. C.; Horrocks, I.; Krötzsch, M.; Kupke, C.; Magka, D.; Motik, B.; and Wang, Z. 2013. Acyclicity Notions for Existential Rules and Their Application to Query Answering in Ontologies. *JAIR* 47:741–808.

Guo, Y.; Pan, Z.; and Heflin, J. 2011. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web* 3(2-3).

König, M.; Leclère, M.; Mugnier, M.; and Thomazo, M. 2015. Sound, complete and minimal ucq-rewriting for existential rules. *Semantic Web* 6(5):451–475.

Krötzsch, M., and Rudolph, S. 2011. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In *IJCAI*.

Leone, N.; Manna, M.; Terracina, G.; and Veltri, P. 2012. Efficiently Computable Datalog$^{\exists}$ Programs. In *KR*.

Marnette, B. 2009. Generalized schema-mappings: from termination to tractability. In *PODS*.

Motik, B.; Nenov, Y.; Piro, R.; Horrocks, I.; and Olteanu, D. 2014. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In *AAAI*.

Motik, B.; Nenov, Y.; Piro, R.; and Horrocks, I. 2015. Handling owl:sameAs via Rewriting. In *AAAI*.

ten Cate, B.; Chiticariu, L.; Kolaitis, P. G.; and Tan, W. C. 2009. Laconic Schema Mappings: Computing the Core with SQL Queries. *PVLDB* 2(1):1006–1017.