Rewriting the Infinite Chase

Michael Benedikt, Maxime Buron, Stefano Germano, Kevin Kappelmann, Boris Motik

HAL Id: lirmm-03714595

https://hal-lirmm.ccsd.cnrs.fr/lirmm-03714595v1

Submitted on 5 Jul 2022

Rewriting the Infinite Chase

Michael Benedikt
Oxford University
Oxford, United Kingdom
michael.benedikt@cs.ox.ac.uk

Maxime Buron
LIRMM, Inria, Univ. of Montpellier
Montpellier, France
maxime.buron@inria.fr

Stefano Germano
Oxford University
Oxford, United Kingdom
stefano.germano@cs.ox.ac.uk

Kevin Kappelmann
Technical University of Munich
Munich, Germany
kevin.kappelmann@tum.de

Boris Motik
Oxford University
Oxford, United Kingdom
boris.motik@cs.ox.ac.uk

ABSTRACT

Guarded tuple-generating dependencies (GTGDs) are a natural extension of description logics and referential constraints. It has long been known that queries over GTGDs can be answered by a variant of the *chase*—a quintessential technique for reasoning with dependencies. However, there has been little work on concrete algorithms and even less on implementation. To address this gap, we revisit *Datalog rewriting* approaches to query answering, where GTGDs are transformed to a Datalog program that entails the same base facts on each base instance. We show that the rewriting can be seen as containing "shortcut" rules that circumvent certain chase steps, we present several algorithms that compute the rewriting by simulating specific types of chase steps, and we discuss important implementation issues. Finally, we show empirically that our techniques can process complex GTGDs derived from synthetic and real benchmarks and are thus suitable for practical use.

1 INTRODUCTION

Tuple-generating dependencies (TGDs) are a natural extension of description logics and referential constraints, and they are extensively used in databases. For example, they are used in data integration to capture semantic restrictions on data sources, mapping rules between data sources and the mediated schema, and constraints on the mediated schema. A fundamental computational problem in such applications is *query answering under TGDs*: given a query Q, a collection of facts I, and a set of TGDs Σ, find all the answers to Q that logically follow from I and Σ. This problem has long been seen as a key component of a declarative data integration systems

[25, 33], and it also arises in answering querying using views and accessing data sources with restrictions [20, 26, 37].

The *chase* is a quintessential technique for reasoning with TGDs. It essentially performs "forward reasoning" by extending a set of given facts I to a set I' of all facts implied by I and a set of TGDs Σ. To answer a query, one can compute I' using the chase and then evaluate the query in I'. Unfortunately, the chase does not necessarily terminate, and in fact query answering for general TGDs is undecidable. Considerable effort was devoted to identifying classes of TGDs for which query answering is decidable. One line of work has focused on TGDs where the chase terminates; *weakly-acyclic TGDs* [21] are perhaps the best-known such class. Another line of work focused on *guarded TGDs (GTGDs)*. GTGDs are interesting since they can capture common constraints used in data integration, and ontologies expressed in variants of *description logic (DL)* [6] can be translated directly into GTGDs. Example 1.1 illustrates the use of GTGDs used in a practical data integration scenario.

EXAMPLE 1.1. The IEC Common Information Model (CIM) is an open model for describing power generation and distribution networks. It is frequently used as a semantic layer in applications that integrate data about power systems [22]. CIM is defined in UML, but its formal semantics has been provided by a translation into an OWL ontology. The domain of CIM is described using *classes* and *properties*, which correspond to unary and binary relations, respectively. Moreover, semantic relationships between classes and properties are represented as OWL axioms, many of which can be translated into GTGDs. A significant portion of CIM describes power distribution equipment using GTGDs such as (1)–(4).

$$\text{ACEquipment}(x) \rightarrow \exists y \, \text{hasTerminal}(x, y) \land \text{ACTerminal}(y) \quad (1)$$
$$\text{ACTerminal}(x) \rightarrow \text{Terminal}(x) \quad (2)$$
$$\text{hasTerminal}(x, z) \land \text{Terminal}(z) \rightarrow \text{Equipment}(x) \quad (3)$$
$$\text{ACTerminal}(x) \rightarrow \exists y \, \text{partOf}(x, y) \land \text{ACEquipment}(y) \quad (4)$$

Data integration is then achieved by populating the vocabulary using *mappings*, which can be seen queries over the data sources that produce a set of facts called a *base instance*. A key issue in data integration is dealing with incompleteness of data sources. For example, it is not uncommon that one data source mentions two switches sw_1 and sw_2, while another data source provides information about connected terminals only for switch sw_1.

$$\text{ACEquipment}(\text{sw}_1) \quad \text{ACEquipment}(\text{sw}_2) \quad (5)$$
$$\text{hasTerminal}(\text{sw}_1, \text{trm}_1) \quad \text{ACTerminal}(\text{trm}_1) \quad (6)$$

GTGDs can be used to complete the data. For example, if a user asks to list all pieces of equipment known to the system, both sw_1 and sw_2 will be returned, even though the base instance does not explicitly classify either switch as a piece of equipment. ◁

Even though the chase for GTGDs does not necessarily terminate, query answering for GTGDs is decidable [34]. To prove decidability, one can argue that the result of a chase is *tree-like*—that is, the facts derived by the chase can be arranged into a particular kind of tree. Next, one can develop a *finite representation* of potentially infinite trees. One possibility is to describe the trees using a *finite tree automaton*, so query answering can be reduced to checking automaton emptiness. While theoretically elegant, this method is not amenable to practical use: building the automaton and the emptiness test are both complex and expensive, and such algorithms always exhibit worst-case complexity. Alternatively, one can use *blocking* to identify a tree prefix sufficient for query evaluation. Blocking is commonly used in description logic reasoning [6], and it was later lifted to *guarded logic* [27]. However, blocking was shown to be impractical for query answering: the required tree prefix can be much larger than the base instance I so, as I grows in size, the size of the tree prefix becomes unmanageable.

More promising query answering techniques for GTGDs are based on *Datalog rewriting* [35]. The idea was initially proposed by Marnette [35], and it was later extended to broader classes of TGDs [10, 24] and settings [11]. The main idea is to transform an input set of GTGDs Σ into a set $\text{rew}(\Sigma)$ of *Datalog* rules such that Σ and $\text{rew}(\Sigma)$ entail the same base facts on each base instance. Thus, given a base instance I, instead of computing the chase of I and Σ (which may not terminate), we compute the chase I' of I and $\text{rew}(\Sigma)$. Since Datalog rules essentially correspond to existential-free TGDs, I' is always finite and it can be computed using optimized Datalog engines. Moreover, Σ and $\text{rew}(\Sigma)$ entail the same base facts on I, so we can answer any *existential-free* conjunctive query (i.e., queries where all variables are answer variables) by evaluating in I'. The restriction to existential-free queries is technical: existentially quantified variables in a query can be matched to objects introduced by existential quantification, and these are not preserved in a Datalog rewriting. However, practical queries are typically existential-free since all query variables are usually answer variables.

EXAMPLE 1.2. A Datalog program consisting of rules (2)–(3) and (7) is a rewriting of GTGDs (1)–(4).

$$\text{ACEquipment}(x) \rightarrow \text{Equipment}(x) \tag{7}$$

Rule (7) is a logical consequence of GTGDs (1)–(3), and it provides a "shortcut" for the inferences of the other GTGDs. ◁

The advantage of rewriting-based approaches is scalability in the size of the base instance I. Such techniques have been implemented and practically validated in the context of description logics [28, 29], but practical algorithms have not yet been proposed for GTGDs. This raises several theoretical and practical questions.

How to compute the Datalog rules needed for completeness? Existing Datalog rewriting algorithms often prove their correctness indirectly. For example, completeness of a rewriting algorithm for description logics [29] uses a proof-theoretic argument, which does not provide an intuition about why the algorithm actually works.

Our first contribution is to *relate Datalog rewriting approaches to the chase*. Towards this goal, we introduce the *one-pass* variant of the chase, which we use to develop a general completeness criterion for Datalog rewriting algorithms. This, in turn, provides us with a better understanding of how rewriting algorithms work, and it allows us to discover new algorithms in a systematic way.

What does the space of rewriting algorithms look like? Computing the rewriting $\text{rew}(\Sigma)$ usually requires extending Σ with certain logical consequences of Σ. We show that we can select the relevant consequences using different criteria. Some methods require deriving TGDs with existential quantifiers in the head, others generate Datalog rules directly, and yet other methods derive logical implications with function symbols. We relate all of these methods to the one-pass chase mentioned earlier, and we provide theoretical worst-case guarantees about their performance.

How do we ensure scalability of rewriting algorithms? Implementations of Datalog rewriting algorithms have thus far been mainly considered in the setting of description logics [29, 38]. *To the best of our knowledge, we provide the first look at optimization and implementation of Datalog rewriting algorithms for GTGDs.* We achieve scalability by developing and combining various indexing and redundancy elimination techniques.

How do we evaluate rewriting algorithms? We provide a benchmark for GTGD query answering algorithms, and we use it to evaluate our methods. To the best of our knowledge, this is the first attempt to evaluate query answering techniques for GTGDs.

Summary of contributions. We give an extensive account of Datalog rewriting for GTGDs. In particular, we develop a theoretical framework that allows us to understand, motivate, and show completeness of rewriting algorithms. Moreover, we present several concrete algorithms, establish worst-case complexity bounds, and discuss their relationships. We complement this theoretical analysis with a discussion of how to adapt techniques from first-order theorem proving to the setting of GTGDs. Finally, we empirically evaluate our techniques using an extensive benchmark. All proofs and the details of one algorithm are given in the appendix of this paper. Our implementation and a more detailed account of our experimental results can be found online [13].

2 RELATED WORK

Answering queries via rewriting has been extensively considered in description logics. For example, queries over ontologies in the DL-Lite family of languages can be rewritten into first-order queries [18], and fact entailment for $\mathcal{SHIQ}$ ontologies can be rewritten to disjunctive Datalog [29]. These techniques provide the foundation for the Ontop [17] and KAON2 [38] systems, respectively.

In the context of TGDs, first-order rewritings were considered in data integration systems with inclusion and key dependencies [16]. Datalog rewritings have been considered for GTGDs [35] and their extensions such as frontier-guarded TGDs [11], and nearly frontier-guarded and nearly guarded TGDs [24]. The focus in these studies was to identify complexity bounds and characterize expressivity of TGD classes rather than provide practical algorithms. Existing implements of query answering for TGDs use first-order rewriting for linear TGDs [48], chase variants for TGDs with terminating chase

[14], chase with blocking for warded TGDs [12], chase with the magic sets transformation for shy TGDs [3], and Datalog rewriting for separable and weakly separable TGDs [49]. These TGD classes are all different from GTGDs, and we are unaware of any attempts to implement and evaluate GTGD rewriting algorithms.

Our algorithms are related to resolution-based decision procedures for variants of guarded logics [19, 23, 50]. Moreover, our characterization of Datalog rewritings is related to a chase variant used to answer queries over data sources with access patterns [4]. Finally, a variant of the one-pass chase from Section 4 was generalized to the broader context of disjunctive GTGDs [31].

3 PRELIMINARIES

In this section, we recapitulate the well-known definitions and notation that we use to formalize our technical results.

TGDs. Let consts, vars, and nulls be pairwise disjoint, infinite sets of *constants*, *variables*, and *labeled nulls*, respectively. A *term* is a constant, a variable, or a labeled null; moreover, a term is *ground* if it does not contain a variable. For α a formula or a set thereof, consts(α), vars(α), nulls(α), and terms(α) are the sets of constants, free variables, labeled nulls, and terms, respectively, in α.

A *schema* is a set of relations, each of which is associated with a nonnegative integer arity. A *fact* is an expression of the form $R(\vec{t})$, where R is an n-ary relation and $\vec{t}$ is a vector of n ground terms; moreover, $R(\vec{t})$ is a *base fact* if $\vec{t}$ contains only constants. An *instance* I is a finite set of facts, and I is a *base instance* if it contains only base facts. An *atom* is an expression of the form $R(\vec{t})$, where R is an n-ary relation and $\vec{t}$ is a vector of n terms not containing labeled nulls. Thus, each base fact is an atom. We often treat conjunctions as sets of conjuncts; for example, for γ a conjunction of facts and I an instance, $\gamma \subseteq I$ means that each conjunct of γ is contained I.

A *tuple generating dependency* (TGD) is a first-order formula of the form $\forall \vec{x}[\beta \to \exists \vec{y}\, \eta]$, where β and η are conjunctions of atoms, η is not empty, the free variables of β are $\vec{x}$, and the free variables of η are contained in $\vec{x} \cup \vec{y}$. Conjunction β is the *body* and formula $\exists \vec{y}\, \eta$ is the *head* of the TGD. We often omit $\forall \vec{x}$ when writing a TGD. A TGD is *full* if $\vec{y}$ is empty; otherwise, the TGD is *non-full*. A TGD is in *head-normal form* if it is full and its head contains exactly one atom, or it is non-full and each head atom contains at least one existentially quantified variable. Each TGD can be easily transformed to an equivalent set of TGDs in head-normal form. A full TGD in head-normal form is a *Datalog rule*, and a *Datalog program* is a finite set of Datalog rules. The *head-width* (hwidth) and the *body-width* (bwidth) of a TGD are the numbers of variables in the head and body, respectively; these are extended to sets of TGDs by taking the maxima over all TGDs. The notion of an instance satisfying a TGD is inherited from first-order logic. A base fact F is *entailed* by an instance I and a finite set of TGDs Σ, written $I, \Sigma \models F$, if $F \in I'$ holds for each instance $I' \supseteq I$ that satisfies Σ.

A *substitution* σ is a function that maps finitely many variables to terms. The domain and the range of σ are dom(σ) and rng(σ), respectively. For γ a term, a vector of terms, or a formula, $\sigma(\gamma)$ is obtained by replacing each free occurrence of a variable x in γ such that $x \in$ dom(σ) with $\sigma(x)$.

Fact Entailment for Guarded TGDs. Fact entailment for general TGDs is semidecidable, and many variants of the *chase* can be used

to define a (possibly infinite) set of facts that is homomorphically contained in each modef of a base instance and a set of TGDs.

Fact entailment is decidable for *guarded* TGDs (GTGDs): a TGD $\forall \vec{x}[\beta \to \exists \vec{y}\, \eta]$ is guarded if β contains an atom (called a *guard*) that contains all variables of $\vec{x}$. Note that a guard need not be unique in β. Let Σ be a finite set of GTGDs. We say that a set of ground terms G is Σ-*guarded* by a fact $R(\vec{t})$ if $G \subseteq \vec{t} \cup$ consts(Σ). Moreover, G is Σ-*guarded* by a set of facts I if G is Σ-guarded by some fact in I. Finally, a fact $S(\vec{u})$ is Σ-*guarded* by a fact $R(\vec{t})$ (respectively a set of facts I) if $\vec{u}$ is Σ-guarded by $R(\vec{t})$ (respectively I).

By adapting the reasoning techniques for guarded logics [5, 47] and referential database constraints [30], fact entailment for GTGDs can be decided by a chase variant that works on tree-like structures. A *chase tree* T consists of a directed tree, one tree vertex that is said to be *recently updated*, and a function mapping each vertex v in the tree to a finite set of facts $T(v)$. A chase tree T can be transformed to another chase tree T' in the following two ways.

- One can apply a *chase step* with a GTGD $\tau = \forall \vec{x}[\beta \to \exists \vec{y}\, \eta]$ in head-normal form. The precondition is that there exist a vertex v in T and a substitution σ with domain $\vec{x}$ such that $\sigma(\beta) \subseteq T(v)$. The result of the chase step is obtained as follows.
  - If τ is full (and thus η is a single atom), then chase tree T' is obtained from T by making v recently updated in T' and setting $T'(v) = T(v) \cup \{\sigma(\eta)\}$.
  - If τ is not full, then σ is extended to a substitution σ' that maps each variable in $\vec{y}$ to a labeled null not occurring in T, and chase tree T' is obtained from T by introducing a fresh child v' of v, making v' recently updated in T', and setting $T(v') = \sigma'(\eta) \cup \{F \in T(v) \mid F \text{ is } \Sigma\text{-guarded by } \sigma'(\eta)\}$.
- One can apply a *propagation step* from a vertex v to a vertex v' in T. Chase tree T' is obtained from T by making v' recently updated in T' and setting $T'(v') = T(v') \cup S$ for some nonempty set S satisfying $S \subseteq \{F \in T(v) \mid F \text{ is } \Sigma\text{-guarded by } T(v')\}$.

A *tree-like chase sequence* for a base instance I and a finite set of GTGDs Σ in head-normal form is a finite sequence of chase trees $T_0, \ldots, T_n$ such that T_0 contains exactly one *root vertex* r that is recently updated in T_0 and $T_0(r) = I$, and each T_i with $0 < i \leq n$ is obtained from T_{i-1} by a chase step with some $\tau \in \Sigma$ or a propagation step. For each vertex v in T_n and each fact $F \in T_n(v)$, this sequence is a *tree-like chase proof of F from I and Σ*. It is well known that $I, \Sigma \models F$ if and only if there exists a tree-like chase proof of F from I and Σ (e.g., [34]). Example 4.3 in Section 4 illustrates these definitions. One can decide $I, \Sigma \models F$ by imposing an upper bound on the size of chase trees that need to be considered [34].

Rewriting. A *Datalog rewriting* of a finite set of TGDs Σ is a Datalog program rew(Σ) such that $I, \Sigma \models F$ if and only if $I, \text{rew}(\Sigma) \models F$ for each base instance I and each base fact F. If Σ contains GTGDs only, then a Datalog rewriting rew(Σ) is guaranteed to exist (which is not the case for general TGDs). Thus, we can reduce fact entailment for GTGDs to Datalog reasoning, which can be solved using highly optimized Datalog techniques [1, 38]. For example, given a base instance I, we can compute the *materialization* of rew(Σ) on I by applying the rules of rew(Σ) to I up to a fixpoint. This will compute precisely all base facts entailed by rew(Σ) (and thus also by Σ) on I, and it can be done in time polynomial in the size of I.

Encoding Existentials by Function Symbols. It is sometimes convenient to represent existentially quantified values using functional terms. In such cases, *we use a slightly modified notions of terms, atoms, and rules*. It will be clear from the context which definitions we use in different parts of the paper.

We adjust the notion of a term as either a constant, a variable, or an expression of the form $f(\vec{t})$ where f is an n-ary *function symbol* and $\vec{t}$ is a vector of n terms. The notions of ground terms, (base) facts, and (base) instances, and atoms are the same as before, but they use the modified notion of terms. A *rule* is a first-order implication of the form $\forall \vec{x}[\beta \rightarrow H]$ where β is a conjunction of atoms whose free variables are $\vec{x}$, and H is an atom whose free variables are contained in $\vec{x}$; as for TGDs, we often omit $\forall \vec{x}$. A rule thus contains no existential quantifiers, but its head contains exactly one atom that can contain function symbols. Also, a Datalog rule, a function-free rule, and a full TGD in head-normal form are all synonyms. Finally, a base fact still contains only constants.

Skolemization allow us to replace existential quantifiers in TGDs by functional terms. Specifically, let $\tau = \forall \vec{x}[\beta \rightarrow \exists \vec{y} \, \eta]$, and let σ be a substitution defined on each $y \in \vec{y}$ as $\sigma(y) = f_{\tau,y}(\vec{x})$ where $f_{\tau,y}$ is a fresh $|\vec{x}|$-ary *Skolem* symbol uniquely associated with τ and y. Then, the Skolemization of τ produces rules $\forall \vec{x}[\beta \rightarrow \sigma(H)]$ for each atom $H \in \eta$. Moreover, the Skolemization Σ' of a finite set of TGDs Σ is the union of the rules obtained by Skolemizing each $\tau \in \Sigma$. It is well known that $I, \Sigma \models F$ if and only if $I, \Sigma' \models F$ for each base instance I and each base fact F.

Unification. A *unifier* of atoms $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ is a substitution θ such that $\theta(A_i) = \theta(B_i)$ for $1 \leq i \leq n$. Such θ is a *most general unifier* (MGU) if, for each unifier σ of $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$, there exists a substitution ρ such that $\sigma = \rho \circ \theta$ (where $\circ$ is function composition). An MGU is unique up to variable renaming if it exists, and it can be computed in time $O(\sum_{i=1}^{n} |A_i| + |B_i|)$ where $|A_i|$ and $|B_i|$ are the encoding sizes of A_i and B_i [41, 42].

4 CHASE-BASED DATALOG REWRITING

Our objective is to develop rewriting algorithms that can handle complex GTGDs. Each algorithm will derive Datalog rules that provide "shortcuts" in tree-like chase proofs: instead of introducing a child vertex v' using a chase step with a non-full GTGD at vertex v, performing some inferences in v', and then propagating a derived fact F back from v' to v, these "shortcuts" will derive F in one step without having to introduce v'. The main question is how to derive all "shortcuts" necessary for completeness while keeping the number of derivations low. In this section we lay the technical foundations that will allow us to study different strategies for deriving "shortcuts" in Section 5. We show that, instead of considering arbitrary chase proofs, we can restrict our attention to chase proofs that are *one-pass* according to Definition 4.1. Then, we identify the parts of such proofs that we need to be able to circumvent using "shortcuts". Finally, we present sufficient conditions that guarantee completeness of rewriting algorithms. We start by describing formally the structure of tree-like chase proofs.

Definition 4.1. *A tree-like chase sequence $T_0, \ldots, T_n$ for a base instance I and a finite set of GTGDs Σ in head-normal form is one-pass if, for each $0 < i \leq n$, chase tree T_i is obtained by applying one of the following two steps to the recently updated vertex v of T_{i-1}:*
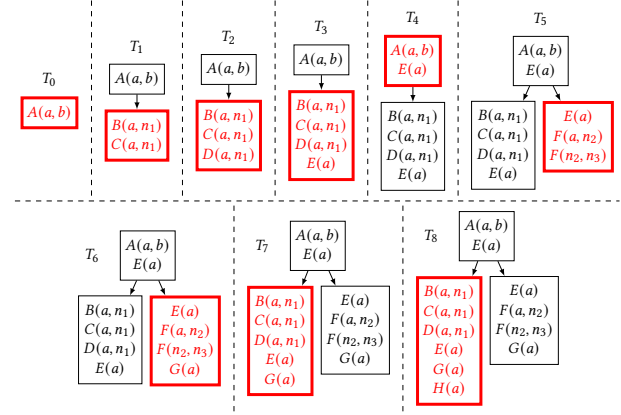

Figure 1: Tree-Like Chase Sequence for Example 4.3

- *a propagation step copying exactly one fact from v to its parent, or*
- *a chase step with a GTGD from Σ provided that no propagation step from v to the parent of v is applicable.*

Thus, each step in a tree-like chase sequence is applied to a "focused" vertex; steps with non-full TGDs move the "focus" from a parent to a child, and propagation steps move the "focus" in the opposite direction. Moreover, once a child-to-parent propagation takes place, the child cannot be revisited in further steps. Theorem 4.2 states a key property about chase proofs for GTGDs: whenever a proof exists, there exists a one-pass proof too. Example 4.3 illustrates important aspects of Definition 4.1 and Theorem 4.2.

THEOREM 4.2. *For each base instance I, each finite set of GTGDs Σ in head-normal form, and each base fact F such that $I, \Sigma \models F$, there exists a one-pass tree-like chase proof of F from I and Σ.*

EXAMPLE 4.3. Let $I = \{A(a, b)\}$ and let Σ contain GTGDs (8)–(13).

$$A(x_1, x_2) \rightarrow \exists y \, B(x_1, y) \wedge C(x_1, y) \tag{8}$$
$$C(x_1, x_2) \rightarrow D(x_1, x_2) \tag{9}$$
$$B(x_1, x_2) \wedge D(x_1, x_2) \rightarrow E(x_1) \tag{10}$$
$$A(x_1, x_2) \wedge E(x_1) \rightarrow \exists y_1, y_2 \, F(x_1, y_1) \wedge F(y_1, y_2) \tag{11}$$
$$E(x_1) \wedge F(x_1, x_2) \rightarrow G(x_1) \tag{12}$$
$$B(x_1, x_2) \wedge G(x_1) \rightarrow H(x_1) \tag{13}$$

A tree-like chase sequence for I and Σ is shown in Figure 1, and it provides a proof of the base fact $H(a)$ from I and Σ. The recently updated vertex of each chase tree is shown in red. We denote the root vertex by r, and its left and right children by v_1 and v_2, respectively. The step producing T_7 from T_6 does *not* satisfy the requirements of one-pass chase: it propagates the fact $G(a)$ from v_2 to v_1, where the latter is a "sibling" of the former.

To obtain a one-pass chase sequence, we could try to "slow down" the propagation of $G(a)$: we first propagate $G(a)$ from v_2 to r, and then from r to v_1. The former step is allowed in one-pass chase, but the latter step is not: once we leave the subtree rooted at v_1, we are not allowed to revisit it later. Note, however, that $B(a, n_1)$ and $G(a)$ must occur jointly in a vertex of a chase tree in order to derive $H(a)$. Moreover, note that no reordering of chase steps will
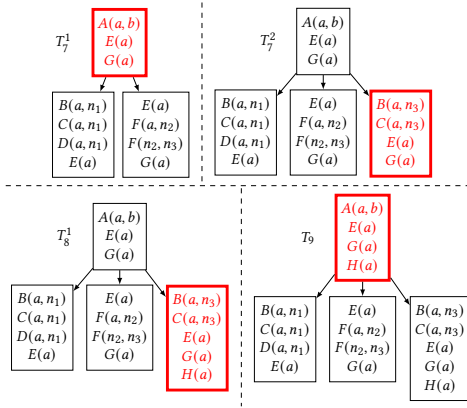
Figure 2: One-Pass Chase Sequence Obtained from Figure 1

derive $H(a)$: we must first produce v_1 to be able to derive v_2, and we must combine $G(a)$ from v_2 and $B(a, n_1)$ from v_1.

The solution, which is used in the proof of Theorem 4.2, is to replace propagation to the child by "regrowing" the entire subtree. In our example, we replace the steps producing T_7 and T_8 with the steps shown in Figure 2. Chase tree T_7^1 is obtained from T_6 by propagating $G(a)$ from v_2 to r. Then, instead of propagating $G(a)$ from r to v_1, a new vertex v_3 is created in T_7^2 by reapplying (8) and fact $G(a)$ is pushed to v_3 as part of the chase step with a non-full GTGD. This allows $H(a)$ to be derived in vertex v_3 of T_8^1.

Fact $D(n_3)$ can be derived in vertex v_3, but this is not needed to prove $H(a)$. Moreover, our chase is *oblivious* [34]: a non-full TGD can be applied to the same facts several times, each time introducing a fresh vertex and fresh labeled nulls. The number of children of a vertex is thus not naturally bounded, and our objective is not to apply all chase steps exhaustively to obtain a universal model of Σ. Instead, we are interested only in *chase proofs*, which must only contain steps needed to demonstrate entailment of a specific fact. ◄

One-pass chase proofs are interesting because they can be decomposed into *loops* as described in Definition 4.4.

Definition 4.4. *For $T_0, \ldots, T_n$ a one-pass tree-like chase sequence for some I and Σ, a* loop *at vertex v with output fact F is a subsequence $T_i, \ldots, T_j$ with $0 \le i < j \le n$ such that*

- T_{i+1} *is obtained by a chase step with a non-full GTGD,*
- T_j *is obtained by a propagation step that copies F, and*
- v *is the recently updated vertex of both T_i and T_j.*

The length *of the loop is defined as $j - i$.*

Example 4.5. Subsequence T_0, T_1, T_2, T_3, T_4 of the chase trees from Example 4.3 is a loop at the root vertex r with output fact $E(a)$: chase tree T_1 is obtained by applying a non-full GTGD to r, and chase tree T_4 is obtained by propagating $E(a)$ back to r. Analogously, T_4, T_5, T_6, T_7^1 is another loop at r with output fact $G(a)$. Finally, T_7^1, T_7^2, T_8^1, T_9 is a loop at r with output fact $H(a)$. ◄

Thus, a loop is a subsequence of chase steps that move the "focus" from a parent to a child vertex, perform a series of inferences in the child and its descendants, and finally propagate one fact back to the parent. If non-full TGDs are applied to the child, then the loop can be recursively decomposed into further loops at the child. The properties of the one-pass chase ensure that each loop is finished as soon as a fact is derived in the child that can be propagated to the parent, and that the vertices introduced in the loop are not revisited at any later point in the proof. In this way, each loop at vertex v can be seen as taking the set $T_i(v)$ as input and producing the output fact F that is added to $T_j(v)$. This leads us to the following idea: for each loop with the input set of facts $T_i(v)$, a rewriting should contain a "shortcut" Datalog rule that derives the loop's output.

Example 4.6. One can readily check that rules (14)–(16) provide "shortcuts" for the three loops identified in Example 4.5.

$$A(x_1, x_2) \rightarrow E(x_1) \tag{14}$$

$$A(x_1, x_2) \wedge E(x_1) \rightarrow G(x_1) \tag{15}$$

$$A(x_1, x_2) \wedge G(x_1) \rightarrow H(x_1) \tag{16}$$

Moreover, these are all relevant "shortcuts": the union of rules (14)–(16) and the Datalog rules from Example 4.3—that is, rules (9), (10), (12), and (13)—is a rewriting of the set Σ from Example 4.1. ◄

These ideas are formalized in Proposition 4.7, which will provide us with a correctness criterion for our algorithms.

Proposition 4.7. *A Datalog program Σ' is a rewriting of a finite set of GTGDs Σ in head-normal form if*

- Σ' *is a logical consequence of Σ,*
- *each Datalog rule of Σ is a logical consequence of Σ', and*
- *for each base instance I, each one-pass tree-like chase sequence $T_0, \ldots, T_n$ for I and Σ, and each loop $T_i, \ldots, T_j$ at the root vertex r with output fact F, there exist a Datalog rule $\beta \rightarrow H \in \Sigma'$ and a substitution σ such that $\sigma(\beta) \subseteq T_i(r)$ and $\sigma(H) = F$.*

Intuitively, the first condition ensures soundness: rewriting Σ' should not derive more facts than Σ. The second condition ensures that Σ' can mimic direct applications of Datalog rules from Σ at the root vertex r. The third condition ensures that Σ' can reproduce the output of each loop at vertex r using a "shortcut" Datalog rule.

5 REWRITING ALGORITHMS

We now consider ways to produce "shortcut" Datalog rules satisfying Proposition 4.7. In Subsection 5.1 we present the ExbDR algorithm that manipulates GTGDs directly, and in Subsections 5.2 and 5.3 we present the SkDR and HypDR algorithms, respectively, that manipulate rules obtained by Skolemizing the input GTGDs. All of these algorithms can produce intermediate GTGDs/rules that are not necessarily Datalog rules. In Appendix E we present the FullDR algorithm that manipulates GTGDs, but derives only Datalog rules. However, the performance of FullDR proved to not be competitive, so we do not discuss it any further here.

Each algorithm is defined by an inference rule Inf that can be applied to several TGDs/rules to derive additional TGDs/rules. For simplicity, we use the same name for the rule and the resulting algorithm. Given a set of GTGDs Σ, the algorithm applies Inf to (the Skolemization of) Σ as long as possible and then returns all produced Datalog rules. This process, however, can derive a large

number of TGDs/rules, so it is vital to eliminate TGDs/rules whenever possible. We next define notions of *redundancy* that can be used to discard certain TGDs/rules produced by Inf.

Definition 5.1. *A TGD $\tau_1 = \forall \vec{x}_1[\beta_1 \rightarrow \exists \vec{y}_1 \; \eta_1]$ is a* syntactic tautology *if it is in head-normal form and $\beta_1 \cap \eta_1 \neq \emptyset$. TGD τ_1 subsumes a TGD $\tau_2 = \forall \vec{x}_2[\beta_2 \rightarrow \exists \vec{y}_2 \; \eta_2]$ if there exists a substitution μ such that $\mathrm{dom}(\mu) = \vec{x}_1 \cup \vec{y}_1$, $\mu(\vec{x}_1) \subseteq \vec{x}_2$, $\mu(\vec{y}_1) \subseteq \vec{y}_1 \cup \vec{y}_2$, $\mu(y) \neq \mu(y')$ for distinct y and y' in $\vec{y}_1$, $\mu(\beta_1) \subseteq \beta_2$, and $\mu(\eta_1) \supseteq \eta_2$.*

A rule $\tau_1 = \forall \vec{x}_1[\beta_1 \rightarrow H_1]$ is a syntactic tautology *if $H_1 \in \beta_1$. Rule τ_1 subsumes a rule $\tau_2 = \forall \vec{x}_2[\beta_2 \rightarrow H_2]$ if there exists a substitution μ such that $\mu(\beta_1) \subseteq \beta_2$ and $\mu(H_1) = H_2$.*

A TGD/rule τ is contained in a set of TGDs/rules S up to redundancy *if τ is a syntactic tautology or some $\tau' \in S$ subsumes τ.*

The following example illustrates Definition 5.1.

EXAMPLE 5.2. Rule $A(x) \wedge B(x) \rightarrow A(x)$ is a syntactic tautology: applying a chase step with it cannot produce a new fact. A non-full TGD in head-normal form cannot be a syntactic tautology since each head atom of such a TGD contains an existentially quantified variable that does not occur in the TGD body.

Rule $\tau_1 = A(f(x_1), f(x_1)) \wedge B(x_1) \rightarrow B(f(x_1))$ is subsumed by rule $\tau_2 = A(x_2, x_3) \rightarrow B(x_2)$ using substitution μ_1 that maps both x_2 and x_3 to $f(x_1)$. If τ_1 derives $B(f(t))$ in one step from a set of facts I by a substitution σ where $\sigma(x_1) = t$, then τ_2 also derives $B(f(t))$ from I in one step by substitution $\sigma \circ \mu_1$. Thus, rule τ_1 is not needed when rule τ_2 is present, so τ_1 can be discarded.

While syntactic tautologies and rule subsumption are standard in first-order theorem proving [8], subsumption of TGDs is more involved. TGD $\tau_3 = A(x_1, x_1) \wedge B(x_1) \rightarrow \exists y_1 \; C(x_1, y_1)$ is subsumed by TGD $\tau_4 = A(x_2, x_3) \rightarrow \exists y_2, y_3 \; C(x_2, y_2) \wedge D(x_3, y_3)$ by substitution μ_2 where $\mu_2(x_2) = \mu_2(x_3) = x_1$, $\mu_2(y_2) = y_1$, and $\mu_2(y_3) = y_3$. The conditions on substitution μ_2 in Definition 5.1 ensure that y_2 and y_3 are not mapped to each other or to x_1. Thus, as in the previous paragraph, the result of each chase step with τ_3 and substitutions σ and σ' can always be obtained (up to isomorphism) by a chase step with τ_4 and substitutions $\sigma \circ \mu_2$ and $\sigma' \circ \mu_2$. ◁

In Definition 5.3 we formalize the notion of applying Inf exhaustively up to redundancy. The definition, however, does not say how to actually do it: we discuss this and other issues in Section 6.

Definition 5.3. *For Inf an inference rule and Σ a finite set of GTGDs, Inf(Σ) is the subset of all Skolem-free Datalog rules of Σ', where Σ' is the smallest set that contains up to redundancy each TGD/rule obtained by*

- *transforming Σ into head-normal form if Inf manipulates TGDs or Skolemizing Σ if Inf manipulates rules, and*
- *selecting an adequate number of premises in Σ', renaming any variables shared by distinct premises, applying Inf to the renamed premises, and transforming the result into head-normal form.*

5.1 The Existential-Based Rewriting

As we discussed in Section 4, each loop $T_i, \ldots, T_j$ at vertex v in a one-pass chase sequence can be seen as taking $T_i(v)$ as input and producing one fact included in $T_j(v)$ as output. Let v' be child of v introduced in T_{i+1}. The idea behind the ExbDR algorithm is to derive all GTGDs such that, for each k with $i < k \leq j$, all facts of $T_k(v')$
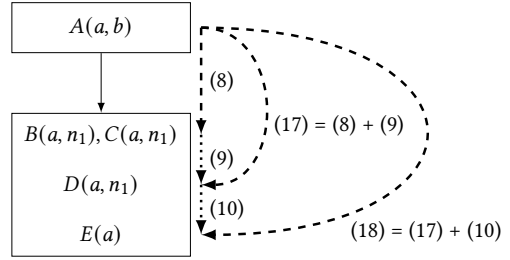

Figure 3: Deriving "shortcuts" for the loop T_0–T_4 in ExbDR

can be derived from the input $T_i(v)$ in one step. The output of the loop can then also be derived from $T_i(v)$ in one step by full GTGD, so this GTGD provides us with the desired loop "shortcut". Before formalizing this idea, we slightly adapt the notion of unification.

Definition 5.4. *For X a set of variables, an X-unifier and an X-MGU θ of atoms $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ are defined as in Section 3, but with the additional requirement that $\theta(x) = x$ for each $x \in X$.*

It is straightforward to see that an X-MGU is unique up to the renaming of variables not contained in X, and that it can be computed as usual while treating variables in X as if they were constants. We are now ready to formalize the ExbDR algorithm.

Definition 5.5. *The* Existential-Based Datalog Rewriting *inference rule* ExbDR *takes two guarded TGDs*

$$\tau = \forall \vec{x}[\beta \rightarrow \exists \vec{y} \; \eta \wedge A_1 \wedge \cdots \wedge A_n] \quad \text{with } n \geq 1 \text{ and}$$
$$\tau' = \forall \vec{z}[A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H']$$

and, for θ a $\vec{y}$-MGU of $A_1, \ldots, A_n$ and $A'_1, \ldots, A'_n$, , if $\theta(\vec{x}) \cap \vec{y} = \emptyset$ and $\mathrm{vars}(\theta(\beta')) \cap \vec{y} = \emptyset$, it derives

$$\theta(\beta) \wedge \theta(\beta') \rightarrow \exists \vec{y} \; \theta(\eta) \wedge \theta(A_1) \wedge \cdots \wedge \theta(A_n) \wedge \theta(H').$$

EXAMPLE 5.6. Consider again the set Σ from Example 4.3. The idea behind the ExbDR algorithm is illustrated in Figure 3, which summarizes the steps of the loop T_0, T_1, T_2, T_3, T_4 from Figure 1. We denote the vertices by r and v_1 as in Example 4.3.

Fact $A(a, b)$ is the input to the loop, and the first step of the loop derives $B(a, n_1)$ and $C(a, n_1)$ using GTGD (8). Next, GTGD (9) evolves vertex v_1 by deriving $D(a, n_1)$. To capture this, the ExbDR inference rule combines (8), the GTGD that creates v_1, with (9), the GTGD that evolves v_1. This produces GTGD (17), which derives all facts of v_1 from the input fact in one step. Vertex v_1 is evolved further using GTGD (10) to derive $E(a)$. To reflect this, the ExbDR inference rule combines (17) and (10) to produce (18), which again derives all facts of v_1 from the loop's input in one step.

$$A(x_1, x_2) \rightarrow \exists y \; B(x_1, y) \wedge C(x_1, y) \wedge D(x_1, y) \tag{17}$$
$$A(x_1, x_2) \rightarrow \exists y \; B(x_1, y) \wedge C(x_1, y) \wedge D(x_1, y) \wedge E(x_1) \tag{18}$$

Fact $E(a)$ does not contain the labeled null n_1 that is introduced when creating v_1, so it can be propagated to the root vertex r as the output of the loop. This is reflected in (18): atom $E(x_1)$ does not contain any existential variables. Definition 5.3 requires each derived GTGD to be brought into head-normal, so (18) is broken up into (17) and (14). The latter GTGD is full, and it provides us with the desired shortcut for the loop.

Next, (12) and atom $F(x_1, y_1)$ of (11) produce (19), and transformation into head-normal form produces (11) and (15). Moreover, (8) and (13) produce (20), and transformation (20) into head-normal form produces (16) and (21).

$$A(x_1, x_2) \land E(x_1) \rightarrow \exists y_1, y_2 \, F(x_1, y_1) \land F(y_1, y_2) \land G(x_1) \quad (19)$$

$$A(x_1, x_2) \land G(x_1) \rightarrow \exists y \, B(x_1, y) \land C(x_1, y) \land H(x_1) \quad (20)$$

$$A(x_1, x_2) \land G(x_1) \rightarrow \exists y \, B(x_1, y) \land C(x_1, y) \quad (21)$$

GTGD (21) is subsumed by (8) so it can be dropped. No further inferences are possible after this, so all derived full GTGDs are returned as the rewriting of Σ. ◂

Before proceeding, we present an auxiliary result showing certain key properties of the ExbDR inference rule.

PROPOSITION 5.7. *Each application of the* ExbDR *inference rule to* τ, τ', *and* θ *as in Definition 5.5 satisfies the following properties.*

1. *Some atom* A_i' *with* $1 \le i \le n$ *is a guard in* τ'.

2. *For each* $1 \le i \le n$ *such that* A_i' *is a guard of* τ', *and for* σ *the* $\vec{y}$-*MGU of* A_i' *and the corresponding atom* A_i *such that* $\sigma(\vec{x}) \cap \vec{y} = \emptyset$, *it is the case that* $\text{vars}(\sigma(A_j')) \cap \vec{y} \ne \emptyset$ *for each* $1 \le j \le n$.

3. *The result is a GTGD whose body and head width are at most* $\text{bwidth}(\Sigma)$ *and* $\text{hwidth}(\Sigma)$, *respectively.*

In the second claim of Proposition 5.7, σ unifies only A_i and A_i', whereas θ unifies all $A_1, \ldots, A_n$ and $A_1', \ldots, A_n'$; thus, σ and θ are not necessarily the same. The third claim is needed to prove termination of ExbDR.

Proposition 5.7 can be used to guide the application of the ExbDR inference rule. Consider an attempt to apply the ExbDR inference rule to two candidate GTGDs $\tau = \beta \rightarrow \exists \vec{y} \, \eta$ and $\tau' = \beta' \rightarrow H'$. The first claim of Proposition 5.7 tells us that a guard of τ' will definitely participate in the inference. Thus, we can choose one such guard $G' \in \beta'$ of τ' and try to find a $\vec{y}$-MGU σ of G' and a counterpart atom $G \in \eta$ from the head of τ. Next, we need to check whether $\sigma(\vec{x}) \cap \vec{y} = \emptyset$; if not, there is no way for $\theta(\vec{x}) \cap \vec{y} = \emptyset$ to hold so the inference is not possible. By the second claim of Proposition 5.7, all candidates for the atoms participating in the inference will contain a variable that is mapped by σ to a member of $\vec{y}$; thus, $S' = \{\sigma(A') \mid A' \in \beta' \land \text{vars}(\sigma(A')) \cap \vec{y} \ne \emptyset\}$ is the set of all relevant side atoms. Note that we apply σ to the atoms in S' to simplify further matching. The next step is to identify the corresponding head atoms of τ. To achieve this, for each atom $A' \in S'$ of the form $R(t_1, \ldots, t_n)$, we identify the set $C[A']$ of candidate counterpart atoms as the set of atoms of the form $R(s_1, \ldots, s_n) \in \sigma(\eta)$ such that, for each argument position i with $1 \le i \le n$, if either $t_i \in \vec{y}$ or $s_i \in \vec{y}$, then $t_i = s_i$. Finally, we consider each possible combination S of such candidates, and we try to find an MGU θ of sets S and S'. If unification succeeds, we derive the corresponding GTGD.

THEOREM 5.8. *Program* ExbDR(Σ) *is a Datalog rewriting of a finite set of GTGDs* Σ. *Moreover, the rewriting can be computed in time* $O(b^{r^d \cdot (w_b + c)^{da}} \cdot r^{d \cdot (w_h + c)^{da}})$ *for* r *the number of relations in* Σ, *a the maximum relation arity in* Σ, $w_b = \text{bwidth}(\Sigma)$, $w_h = \text{hwidth}(\Sigma)$, $c = |\text{consts}(\Sigma)|$, *and some* b *and* d.

Program ExbDR(Σ) can thus be large in the worst case. In Section 7 we show empirically that rewritings are suitable for practical

use. From a theoretical point of view, checking fact entailment via ExbDR(Σ) is worst-case optimal. To see why, let r, a, and c be as in Theorem 5.8, and consider a base instance I with c' constants. The fixpoint of ExbDR(Σ) on I contains at most $r(c + c')^a$ facts, and it can be computed in time $O(r(c + c')^a \cdot |\text{ExbDR}(\Sigma)|)$: each rule $\tau \in \text{ExbDR}(\Sigma)$ is guarded so we can apply a chase step with τ by matching a guard and then checking the remaining body atoms. Hence, we can compute ExbDR(Σ) and find its fixpoint in 2EXP-TIME, in EXPTIME if the relation arity is fixed, and in PTIME if Σ is fixed (i.e., if we consider *data complexity*). These results match the lower bounds for checking fact entailment for GTGDs [34].

5.2 Using Skolemization

The ExbDR algorithm exhibits two drawbacks. First, each application of the ExbDR inference rule potentially introduces a head atom, so the rule heads can get very long. Second, each inference requires matching a subset of body atoms of τ' to a subset of the head atoms of τ; despite the optimizations outlined after Proposition 5.7, this can be costly, particularly when rule heads are long.

We would ideally derive GTGDs with a single head atom and unify just one body atom of τ' with the head atom of τ, but this does not seem possible if we stick to manipulating GTGDs. For example, atoms $C(y)$ and $D(y)$ of GTGD (17) refer to the same labeled null (represented by variable y), and this information would be lost if we split (17) into two GTGDs. We thus need a way to refer to the same existentially quantified object in different logical formulas. This can be achieved by replacing existentially quantified variables by Skolem terms, which in turns gives rise to the SkDR algorithm from Definition 5.10. Before presenting the algorithm, in Definition 5.9 we generalize the notion of guardedness to rules.

Definition 5.9. *Rule* $\forall \vec{x}[\beta \rightarrow H]$ *is guarded if each function symbol in the rule is a Skolem symbol, the body β contains a Skolem-free atom $A \in \beta$ such that* $\text{vars}(A) = \vec{x}$, *and each Skolem term in the rule is of the form $f(\vec{t})$ where $\text{vars}(f(\vec{t})) = \vec{x}$ and $\vec{t}$ is function-free.*

Definition 5.10. *The* Skolem Datalog Rewriting *inference rule* SkDR *takes two guarded rules*

$$\tau = \beta \rightarrow H \qquad and \qquad \tau' = A' \land \beta' \rightarrow H'$$

such that

- β *is Skolem-free and* H *contains a Skolem symbol, and*
- A' *contains a Skolem symbol, or* τ' *is Skolem-free and* A' *contains all variables of* τ',

and, for θ *an MGU of* H *and* A', *it derives*

$$\theta(\beta) \land \theta(\beta') \rightarrow \theta(H').$$

EXAMPLE 5.11. Skolemizing GTGDs (8) and (11) produces rules (22)–(23), and (24)–(25), respectively.

$$A(x_1, x_2) \rightarrow B(x_1, f(x_1, x_2)) \quad (22)$$

$$A(x_1, x_2) \rightarrow C(x_1, f(x_1, x_2)) \quad (23)$$

$$A(x_1, x_2) \land E(x_1) \rightarrow F(x_1, g(x_1, x_2)) \quad (24)$$

$$A(x_1, x_2) \land E(x_1) \rightarrow F(g(x_1, x_2), h(x_1, x_2)) \quad (25)$$

Intuitively, rules (22) and (23) jointly represent the facts introduced by the non-full GTGD (8): functional term $f(x_1, x_2)$ allows both

rules to "talk" about the same labeled nulls. This allows the SkDR inference rule to simulate the ExbDR inference rule while unifying just pairs of atoms. In particular, SkDR combines (22) and (10) to obtain (26); it combines (23) and (9) to obtain (27); and it combines (26) and (27) to obtain the "shortcut" rule (14).

$$A(x_1, x_2) \land D(x_1, f(x_1, x_2)) \to E(x_1) \tag{26}$$

$$A(x_1, x_2) \to D(x_1, f(x_1, x_2)) \tag{27}$$

The rules with Skolem-free bodies derived in this way allow us to reconstruct derivations in one step analogously to Example 5.6, and the rules with Skolem symbols in body atoms capture the intermediate derivation steps. For example, rules (26) and (28) capture the result of matching the first and the second body atom, respectively, of rule (10) to facts produced by rules (22) and (27), respectively. To complete the rewriting, SkDR combines (24) with (12) to obtain (15), and it combines (22) with (13) to derive (16).

However, SkDR also combines (10) and (27) into (28), which with (22) derives (14) the second time. These inferences are superfluous: they just process the two body atoms of (10) in a different order. Also, SkDR combines (12) and (25) into rule (29), which is a "dead-end" in that it does not further contribute to a Datalog rule.

$$A(x_1, x_2) \land B(x_1, f(x_1, x_2)) \to E(x_1) \tag{28}$$

$$A(x_1, x_2) \land E(x_1) \land E(g(x_1, x_2)) \to G(g(x_1, x_2)) \tag{29}$$

Our HypDR algorithm in Subsection 5.3 can avoid these overheads, but at the expense of using more than two rules at a time. ◄

Proposition 5.12 and Theorem 5.13 capture the relevant properties of the SkDR algorithm.

PROPOSITION 5.12. *Each application of the* SkDR *inference rule to rules τ and τ' as in Definition 5.10 produces a guarded rule.*

THEOREM 5.13. *Program* SkDR(Σ) *is a Datalog rewriting of a finite set of GTGDs Σ. Moreover, the rewriting can be computed in time $O(b^{r^d \cdot (e+w_b+c)^{da}})$ for r the number of relations in Σ, a the maximum relation arity in Σ, e the number of existential quantifiers in Σ, $w_b = \text{bwidth}(\Sigma)$, $c = |\text{consts}(\Sigma)|$, and some b and d.*

It is natural to wonder whether SkDR is guaranteed to be more efficient than ExbDR. We next show that neither algorithm is generally better: there exist families of inputs on which SkDR performs exponentially more inferences than ExbDR, and vice versa.

PROPOSITION 5.14. *There exists a family $\{\Sigma_n\}_{n\in\mathbb{N}}$ of finite sets of GTGDs such that the number of GTGDs derived by* ExbDR *is $O(2^n)$ times larger than the number of rules derived by* SkDR *on each Σ_n.*

PROOF. For each $n \in \mathbb{N}$, let Σ_n contain the following GTGDs.

$$A(x) \to \exists \vec{y}\ B_1(x, y_1) \land \cdots \land B_n(x, y_n) \tag{30}$$

$$B_i(x_1, x_2) \land C_i(x_1) \to D_i(x_1, x_2) \text{ for } 1 \le i \le n \tag{31}$$

On such Σ_n, ExbDR derives a GTGD of the form (32) for each subset $\{k_1, \ldots, k_m\} \subseteq \{1, \ldots, n\}$, and there are 2^n such TGDs. In contrast, the Skolemization of (30) consists of n rules shown in equation (33), so SkDR derives just n rules shown in equation (34).

$$A(x) \land \bigwedge_{i=1}^{m} C_{k_i}(x) \to \exists \vec{y} \bigwedge_{i=1}^{n} B_i(x, y_i) \land \bigwedge_{i=1}^{m} D_{k_i}(x, y_{k_i}) \tag{32}$$

$$A(x) \to B_i(x, f_i(x)) \text{ for } 1 \le i \le n \tag{33}$$

$$A(x) \land C_i(x) \to D_i(x, f_i(x)) \text{ for } 1 \le i \le n \qquad \square\tag{34}$$

PROPOSITION 5.15. *There exists a family $\{\Sigma_n\}_{n\in\mathbb{N}}$ of finite sets of GTGDs such that the number of rules derived by* SkDR *is $O(2^n)$ times larger than the number of TGDs derived by* ExbDR *on each Σ_n.*

PROOF. For each $n \in \mathbb{N}$, let Σ_n contain the following GTGDs.

$$A(x) \to \exists y\ B_1(x, y) \land \cdots \land B_n(x, y) \tag{35}$$

$$B_1(x_1, x_2) \land \cdots \land B_n(x_1, x_2) \to C(x_1) \tag{36}$$

On such Σ_n, ExbDR derives just GTGD (37) in one step. In contrast, the Skolemization of (35) consists of n rules of the form (38) for each $1 \le i \le n$. Thus, SkDR combines these with (36) to derive $2^n - 1$ rules of the form (39), one for each subset $\{k_1, \ldots, k_m\} \subsetneq \{1, \ldots, n\}$.

$$A(x) \to C(x) \tag{37}$$

$$A(x) \to B_i(x, f(x)) \tag{38}$$

$$A(x) \land B_{k_1}(x, f(x)) \land \cdots \land B_{k_m}(x, f(x)) \to C(x) \ \square \tag{39}$$

5.3 Combining Several SkDR Steps into One

The SkDR algorithm can produce many rules with Skolem symbols in the body, which is the main reason for Proposition 5.15. We next present the HypDR algorithm, which uses the *hyperresolution* inference rule as a kind of "macro" to combine several SkDR steps into one. We show that this can be beneficial for several reasons.

Definition 5.16. *The* Hyperresolution Rewriting *inference rule* HypDR *takes guarded rules*

$$\tau_1 = \beta_1 \to H_1 \quad \ldots \quad \tau_n = \beta_n \to H_n \text{ and}$$
$$\tau' = A'_1 \land \cdots \land A'_n \land \beta' \to H'$$

such that

- *for each i with $1 \le i \le n$, conjunction β_i is Skolem-free and atom H_i contains a Skolem symbol, and*
- *rule τ' is Skolem-free,*

and, for θ an MGU of $H_1, \ldots, H_n$ and $A'_1, \ldots, A'_n$, if conjunction $\theta(\beta')$ is Skolem-free, it derives

$$\theta(\beta_1) \land \cdots \land \theta(\beta_n) \land \theta(\beta') \to \theta(H').$$

EXAMPLE 5.17. The HypDR inference rule simulates chase steps in the child vertex of a loop analogously to ExbDR: all body atoms matching a fact introduced in the child vertex are resolved in one step. We can see two benefits of this on our running example.

First, HypDR derives (27) from (23) and (9), and it derives (14) from (10), (22), and (27). Rule (14) is derived just once, and without intermediate rules (26) and (28). In other words, the HypDR inference rule does not resolve the body atoms of a rule in every possible order. As Proposition 5.20 below shows, this can reduce the number of derived rules by an exponential factor.

Second, HypDR derives only rules with Skolem-free bodies, and thus does not derive the "dead-end" rule (29). In other words, all consequences of HypDR derive in one step one fact in the child vertex of a loop from the loop's input $T_i(v)$.

The downside of HypDR is that more than two rules can participate in an inference. This requires more complex unification and selection of candidates that can participate in an inference. ◄

Proposition 5.18 and Theorem 5.19 capture the properties of HypDR, and Proposition 5.20 compares it to SkDR.

PROPOSITION 5.18. *Each application of the* HypDR *inference rule to rules* $\tau_1, \ldots, \tau_n$ *and* τ' *as in Definition 5.16 produces a guarded rule.*

THEOREM 5.19. *Program* HypDR(Σ) *is a Datalog rewriting of a finite set of GTGDs* Σ. *Moreover, the rewriting can be computed in time time* $O(b^{r^d \cdot (e+w_b+c)^{da}})$ *for* r *the number of relations in* Σ, *a the maximum relation arity in* Σ, *e the number of existential quantifiers in* Σ, $w_b = \mathrm{bwidth}(\Sigma)$, $c = |\mathrm{consts}(\Sigma)|$, *and some* b *and* d.

PROPOSITION 5.20. *There exists a family* $\{\Sigma_n\}_{n \in \mathbb{N}}$ *of finite sets of GTGDs such that* SkDR *derives* $O(2^n)$ *more rules than* HypDR *on each* Σ_n.

PROOF. For each $n \in \mathbb{N}$, let Σ_n contain the following GTGDs.

$$A(x) \rightarrow \exists y \, B(x, y) \tag{40}$$

$$B(x_1, x_2) \wedge C_i(x_1) \rightarrow D_i(x_1, x_2) \text{ for } 1 \le i \le n \tag{41}$$

$$D_1(x_1, x_2) \wedge \cdots \wedge D_n(x_1, x_2) \rightarrow E(x_1) \tag{42}$$

Skolemizing (40) produces (43). Thus, SkDR combines (43) with each (41) to derive each (44), and it uses (44) and (42) to derive $2^n - 1$ rules of the form (45) for each set of indexes I satisfying $\emptyset \subsetneq I \subseteq \{1, \ldots, n\}$; note that none of these rules are redundant.

$$A(x) \rightarrow B(x, f(x)) \tag{43}$$

$$A(x) \wedge C_i(x) \rightarrow D_i(x, f(x)) \text{ for } 1 \le i \le n \tag{44}$$

$$A(x) \wedge \bigwedge_{i \in I} C_i(x) \wedge \bigwedge_{j \in \{1, \ldots, n\} \setminus I} D_j(x, f(x)) \rightarrow E(x) \tag{45}$$

In contrast, HypDR derives each (44) just like SkDR, and it combines in one step (42) and all (44) to derive (45) for $I = \{1, \ldots, n\}$. □

6 IMPLEMENTATION AND OPTIMIZATIONS

In this section, we discuss numerous issues that have to be addressed to make the computation of a rewriting practical.

Computing Inf(Σ) **in Practice.** Definition 5.3 does not specify how to compute the set Σ', and redundancy elimination makes this question nontrivial. When Inf derives a TGD/rule τ, we can apply subsumption in two ways. First, we can discard τ if τ is subsumed by a previously derived TGD/rule; this is known as *forward subsumption*. Second, if τ is not discarded, we can discard each previously derived TGD/rule that is subsumed by τ; this is known as *backward subsumption*. The set of derived TGD/rules can thus grow and shrink, so the application of Inf has to be carefully structured to ensure that all inferences are performed eventually.

We address this problem by a variant of the *Otter loop* [36] used in first-order theorem provers. The pseudo-code is shown in Algorithm 1. The algorithm maintains two sets of TGDs/rules: the *worked-off* set $\mathcal{W}$ contain TGDs/rules that have been processed by Inf, and the *unprocessed* set $\mathcal{U}$ contains TGDs/rules that are still to be processed. Set $\mathcal{W}$ is initially empty (line 1), and set $\mathcal{U}$ is initialized to the head-normal form of Σ if Inf manipulates TGDs, or to the Skolemization of Σ if Inf manipulates rules. The algorithm then processes each $\tau \in \mathcal{U}$ until $\mathcal{U}$ becomes empty (lines 3–10). It is generally beneficial to process shorter TGDs/rules first as that improves chances of redundancy elimination. After moving τ to $\mathcal{W}$ (line 5),

---

Algorithm 1 Computing Inf(Σ) for Σ a finite set of GTGDs

1: $\mathcal{W} = \emptyset$
2: $\mathcal{U}$ = the head-normal form or the Skolemization of Σ
3: **while** $\mathcal{U} \neq \emptyset$ **do**
4:     Choose some $\tau \in \mathcal{U}$ and remove it from $\mathcal{U}$
5:     $\mathcal{W} = \mathcal{W} \cup \{\tau\}$
6:     Let $\mathcal{E}$ be the result of applying Inf to τ and a subset of $\mathcal{W}$
       and transforming the result into head-normal form
7:     **for each** $\tau' \in \mathcal{E}$ **do**
8:       **if** τ' is not contained in $\mathcal{W} \cup \mathcal{U}$ up to redundancy **then**
9:         Remove from $\mathcal{W}$ and $\mathcal{U}$ each τ'' subsumed by τ'
10:         $\mathcal{U} = \mathcal{U} \cup \{\tau'\}$
11: **return** $\{\tau \in \mathcal{W} \mid \tau$ is a Skolem-free Datalog rule$\}$

---

the algorithm applies Inf to τ and $\mathcal{W}$ and transforms the results into head-normal form (line 6). The algorithm discards each resulting $\tau' \in \mathcal{E}$ that is a syntactic tautology or is forward-subsumed by an element of $\mathcal{W} \cup \mathcal{U}$ (line 8). If τ' is not discarded, the algorithm applies backward subsumption to τ', $\mathcal{W}$, and $\mathcal{U}$ (line 9) and adds τ' to $\mathcal{U}$ (line 10). When all TGDs/rules are processed, the algorithm returns all Skolem-free Datalog rules from $\mathcal{W}$ (line 11). The result of applying Inf to TGDs/rules in $\mathcal{W}$ is thus contained in $\mathcal{W} \cup \mathcal{U}$ up to redundancy at all times so, upon algorithm's termination, set $\mathcal{W}$ satisfies the condition on Σ' from Definition 5.3.

Checking Subsumption. Checking whether TGD/rule τ_1 subsumes τ_2 is NP-complete [32], and the main difficulty is in matching the variables of τ_1 to the variables of τ_2. Thus, we use an approximate check in our implementation. First, we normalize each TGD to use fixed variables $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$: we sort the body and head atoms by their relations using an arbitrary, but fixed ordering and breaking ties arbitrarily, and then we rename all variables so that the ith distinct occurrence of a universally (respectively existentially) quantified variable from left to right is x_i (respectively y_i). To see whether $\tau_1 = \beta_1 \rightarrow \exists \vec{y} \, \eta_1$ subsumes $\tau_2 = \beta_2 \rightarrow \exists \vec{y} \, \eta_2$, we determine whether $\beta_1 \subseteq \beta_2$ and $\eta_1 \supseteq \eta_2$ holds, which requires only polynomial time. We use a similar approximation for rules. Variable normalization ensures termination, and using a modified subsumption check does not affect the correctness of the rewriting: set $\mathcal{W}$ may contain more TGDs/rules than strictly necessary, but these are all logical consequences of (the Skolemization of) Σ.

Subsumption Indexing. Sets $\mathcal{W}$ and $\mathcal{U}$ can be large, so we use a variant of *feature vector indexing* [44] to retrieve subsumption candidates in $\mathcal{W} \cup \mathcal{U}$. For simplicity, we consider only TGDs in the following discussion, but rules can be handled analogously. Note that a TGD τ_1 can subsume TGD τ_2 only if the set of relations occurring in the body of τ_1 (respectively the head of τ_2) is a subset of the set of relations occurring in the body of τ_2 (respectively the head of τ_1). Thus, we can reduce the problem of retrieving subsumption candidates to the problem of, given a domain set D, a set N of subsets of D, a subset $S \subseteq D$, and $\bowtie \in \{\subseteq, \supseteq\}$, retrieving each $S' \in N$ satisfying $S' \bowtie S$. The set-trie data structure [43] can address this problem. The idea is to order D in an arbitrary, yet fixed way, so that we can treat each subset of N as a word over D. We then index N by constructing a trie over the words representing the elements

of N. Finally, we retrieve all $S' \in N$ satisfying $S' \bowtie S$ by traversing the trie, where the ordering on D allows us to considerably reduce the number of vertices we visit during the traversal.

A minor issue is that retrieving TGDs that subsume a given TGD requires both subset and superset testing for body and head relations, respectively, and vice versa for retrieval of subsumed TGDs. To address this, we introduce a distinct symbol R^b and R^h for each relation R occurring in Σ, and we represent each TGD τ as a *feature vector* F_τ of these symbols corresponding to the body and head of τ. Moreover, we combine in the obvious way the subset and superset retrieval algorithms. For example, when searching for a TGD $\tau' \in \mathcal{W} \cup \mathcal{U}$ that subsumes a given TGD τ, we use the subset retrieval for the symbols R^b and the superset retrieval for symbols R^h. Finally, we order these symbols by the decreasing frequency of the order of the symbols' occurrence in the set Σ of input TGDs, and moreover we order each R^b before all R^h.

Relation Clustering. We observed that the subsumption indexes can easily get very large, so index traversal can become a considerable source of overhead. To reduce the index size, we group the symbols R^b and R^h into clusters C^b and C^h, respectively. Then, the feature vector F_τ associated with each TGD τ consists of all clusters C^b and C^h that contain a relation occurring in the body and head, respectively, of τ. We adapt the trie traversal algorithms in the obvious way to take into account this change. The number of clusters is computed using the average numbers of symbols and atoms in the input TGDs, and clusters are computed with the aim of balancing the number of TGDs stored in each leaf vertex.

Unification Indexing. We construct indexes over $\mathcal{W}$ that allow us to quickly identify TGDs/rules that can participate in an inference with some τ. For TGDs, we maintain a hash table that maps each relation R to a set of TGDs containing R in the body, and another hash table that does the same but for TGD heads. To index rules, we use a variant of a *path indexing* [45]: each atom in a rule is represented as a sequence of relations and function symbols occurring in the atom, and such sequences are entered into two tries (one for body and one for head atoms). Then, given rule τ, we consider each body and head atom A of τ, we convert A into the corresponding sequence, and we use the sequence to query the relevant trie for all candidates participating in an inference with τ on A.

Cheap Lookahead Optimization. Consider an application of the ExbDR inference rule to GTGDs τ and τ' as in Definition 5.5, producing a GTGD τ'' where $vars(\theta(H')) \cap \vec{y} \neq \emptyset$ and the relation of H' does not occur in the body of a GTGD in Σ. In each one-pass chase sequence for some base instance and Σ, no GTGD of Σ can be applied to a fact obtained by instantiating $\theta(H')$, so deriving this fact is redundant. Consequently, we can drop such τ'' as soon as we derive it in line 6. Analogously, when the SkDR inference rule is applied to rules τ and τ' as in Definition 5.10, we can drop the resulting rule if $\theta(H')$ is not full and it contains a relation not occurring in the body of a GTGD in Σ.

7 EXPERIMENTAL EVALUATION

We implemented a system that can produce a Datalog rewriting of a set of GTGDs using our algorithms, and we conducted an empirical evaluation using a comprehensive collection of 428 synthetic and realistic inputs. Our objectives were to show that our algorithms can indeed rewrite complex GTGDs, and that the rewriting can be successfully processed by modern Datalog systems. In Subsection 7.1 we describe the test setting. Then, in Subsection 7.2 we discuss the rewriting experiments with GTGDs obtained from ontologies, and in Subsection 7.3 we validate the usefulness of the rewriting approach end-to-end. Finally, in Subsection 7.4 we discuss rewriting GTGDs of higher arity. Due to the very large number of inputs, we can only summarize our results in this paper; however, our complete evaluation results are available online [13].

Table 1: Input GTGDs at a Glance

Inputs	# Full TGDs				# Non-Full TGDs			
	Min	Max	Avg	Med	Min	Max	Avg	Med
428	1	171,905	11,030	789	2	156,743	5,255	283

### 7.1 Input GTGDs, Competitors, & Test Setting

Before discussing our results, we next describe our test setting.

Input GTGDs. We are unaware of any publicly available sets of GTGDs that we could readily use in our evaluation, so we derived the input GTGDs for our evaluation from the ontologies in the Oxford Ontology Library [40]. At the time of writing, this library contained 787 ontologies, each assigned a unique five-digit identifier. After removing closely-related ontology variants, we were left with 428 core ontologies. We loaded each ontology using the parser from the Graal system [9], discarded axioms that cannot be translated into GTGDs, and converted the remaining axioms into GTGDs. We used the standard translation of description logics into first-order logic [6], where each class corresponds to a unary relation, and each property corresponds to a binary relation. We thus obtained 428 sets of input GTGDs with properties shown in Table 1.

To evaluate our algorithms on TGDs containing relations of arity higher than two, we devised a way to "blow up" relation arity. Given a set of GTGDs and a blowup factor b, our method proceeds as follows. First, in each atom of each GTGD, it replaces each variable argument with b fresh variables uniquely associated with the variable; for example, for $b = 2$, atom $A(x, y)$ is transformed into atom $A(x_1, x_2, y_1, y_2)$. Next, the method randomly introduces fresh head and body atoms over the newly introduced variables; in doing so, it ensures that the new atoms do not introduce patterns that would prevent application of the ExbDR inference rule.

Competitors. We compared the ExbDR, SkDR, and HypDR algorithms, implemented as described in Section 6. As noted in Section 2, no existing system we are aware of implements a Datalog rewriting algorithm for GTGDs. However, the KAON2 system [29, 38, 39] can rewrite GTGDs obtained from OWL ontologies, so we used KAON2 as a baseline in our experiments with OWL-based GTGDs. We made sure that all inputs to KAON2 and our algorithms include only GTGDs that all methods can process.

Test Setting. We conducted all experiments on a laptop with an Intel Core i5-6500 CPU @ 3.20 GHz and 16 GB of RAM, running Ubuntu 20.04.4 LTS and Java 11.0.15. In each test run, we loaded a set of TGDs, measured the wall-clock time required to compute
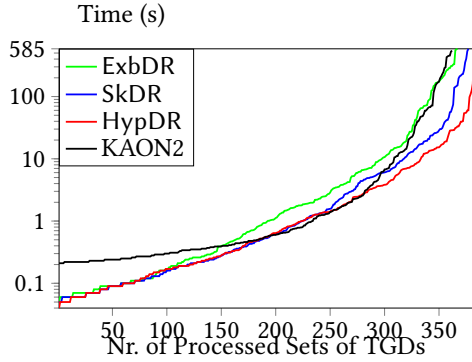
Time (s)

	ExbDR	SkDR	HypDR	KAON2
# of Processed Inputs	367	377	382	362
Max. Processed Input Size	185,515	324,092	324,092	N/A
Max. Output Size	196,594	124,846	124,846	61,964
Max. Size Blowup	8.95	8.85	8.85	N/A
Max. Body Atoms in Output	7	6	6	4
# Blowup ≥ 1.5	26	14	16	N/A
Time (s) Min.	0.05	0.05	0.04	0.21
Max.	582.18	584.79	404.34	547.53
Avg.	23.23	14.34	6.38	18.66
Med.	0.82	0.52	0.55	0.49

	$time(Y)/time(X) \geq 10$				X and Y both fail			
X \ Y	ExbDR	SkDR	HypDR	KAON2	ExbDR	SkDR	HypDR	KAON2
ExbDR		19	0	19	61			
SkDR	37		0	26	33	51		
HypDR	37	12		31	35	43	46	
KAON2	35	15	0		37	47	46	66

Figure 4: Results for TGDs Derived from Ontologies

the rewriting of a set of GTGDs, and saved the produced Datalog rewriting. We used a timeout of ten minutes for each test run.

7.2 Experiments with GTGDs from Ontologies

We computed the Datalog rewriting of GTGDs obtained from OWL ontologies using our three algorithms and KAON2. Figure 4 shows the number of inputs that each algorithm processed in a given time, provides information about the inputs and outputs of each system, and compares the performance among systems. The input size for ExbDR is the number of GTGDs after transforming the input into head-normal form, and for SkDR and HypDR it is the number of rules after Skolemization. Input size is not available for KAON2 since this system reads an OWL ontology and transforms it into GTGDs internally. The output size is the number of Datalog rules in the rewriting. Finally, the blowup is the ratio of the output and the input sizes. Each input GTGD contained at most seven body atoms. Out of 428 inputs, 349 were processed within the ten minute limit by our three systems, and 334 inputs were processed by all four systems. Moreover, 32 inputs, each containing between 20,270 and 221,648 GTGD, were not processed by any system.

Discussion. As one can see in Figure 4, all algorithms were able to compute the rewriting of large inputs containing 100k+ GTGDs. Moreover, for the vast majority of inputs that were successfully

processed, the size of the rewriting and the number of body atoms in the rewriting are typically of the same order of magnitude as the input. Hence, the worst-case exponential blowup from Theorems 5.8, 5.13, and 5.19 does not appear in practice: the size of the rewriting seems to be determined primarily by the input size.

Relative Performance. No system can be identified as the best in general, but HypDR seems to offer the best performance on average. The algorithm was able to process most inputs; it was at least 35% faster than the other systems on the slowest input; it was never slower by an order of magnitude; there were only 14 inputs that could be processed by some other algorithm but not HypDR; and the output of HypDR does not differ significantly from the output of SkDR. This is in line with our motivation for HypDR outlined in Example 5.17. Specifically, HypDR derives rules with just one head atom, but it does not derive intermediate rules with functional body atoms. The main source of overhead in HypDR seems to be more complex selection of rules participating in an inference.

Impact of Subsumption. All algorithms spend a considerable portion of their running time checking TGD/rule subsumption, so it is natural to wonder whether this overhead is justified. To answer this question, we ran our three approaches using a modification of Algorithm 1: we replaced the check for containment up to redundancy in line 8 with just checking $\tau' \notin \mathcal{W} \cup \mathcal{U}$, and we removed line 9. Note that our normalization of variables described in Section 6 still guarantees termination. This change significantly increased the number of derivations: the numbers of derived TGDs/rules increased on average by a factor of 104, 185, and 103 on ExbDR, SkDR, and HypDR, respectively. Interestingly, this increase did not affect the performance uniformly. While SkDR was able to process 12 inputs an order of magnitude faster, ExbDR and HypDR timed out on 72 and 17 additional inputs, respectively. This, we believe, is due to how different inference rules select inference candidates. The SkDR rule is applied to just pairs of rules, and candidate pairs can be efficiently retrieved using unification indexes. In contrast, ExbDR requires matching several head atoms with as many body atoms, which makes developing a precise index for candidate pair retrieval difficult; thus, as the number of derived TGDs increases, the number of false candidates retrieved from the index increases as well. Finally, HypDR can be applied to an arbitrary number of rules, so selecting inference candidates clearly becomes more difficult as the number of derived rules increases.

Impact of Structural Transformation. KAON2 uses *structural transformation* [7] to simplify ontology axioms before translating them into GTGDs. For example, axiom $A \sqsubseteq \exists B.\exists C.D$ is transformed into $A \sqsubseteq \exists B.X$ and $X \sqsubseteq \exists C.D$ for X a fresh class. The resulting axioms have simpler structure, which is often beneficial to performance. To see how this transformation affects our algorithms, we reran our experiments while transforming the input axioms in the same way as in KAON2. This indeed improved the performance of SkDR by one order of magnitude on 22 ontologies, and it did not hurt the performance of HypDR. The main challenge is to generalize this transformation to arbitrary GTGDs: whereas description logic axioms exhibit syntactic nesting that lends itself naturally to this transformation, it is less clear how to systematically apply this

Table 2: Computing the Fixpoint of the Rewriting

Ont. ID	# Rules	# Input Facts	# Output Facts	Time (s)
00387	63,422	4,403,105	51,439,424	53
00448	67,986	5,510,444	107,235,697	110
00470	75,146	10,532,943	141,396,446	242
00471	78,977	11,077,423	128,954,126	253
00472	75,146	10,533,008	141,396,576	279
00473	78,977	11,077,459	128,954,198	291
00573	113,959	9,197,254	155,118,592	206
00682	68,461	5,183,460	105,431,952	101
00684	81,553	6,057,017	66,981,628	109
00686	124,846	10,402,324	166,366,039	238

transformation to TGDs, where heads and bodies consist of "flat" conjunctions. We leave this question for future work.

7.3 End-to-End Experiments

To validate our approach end-to-end, we selected ten inputs where ExbDR produced the largest rewritings. For each of these, we generated a large base instance using WatDiv [2], and we computed the fixpoint of the rewriting and the instance using the RDFox [46] Datalog system v5.4. Table 2 summarizes our results.

All programs used in this experiment are at least several orders of magnitude larger than what is usually encountered in practical applications of Datalog, but RDFox nevertheless computed the fixpoint of all rewritings in a few minutes. Moreover, although the fixpoints seem to be an order of magnitude larger than the base instance, this is not a problem for highly optimized systems such as RDFox. Hence, checking fact entailment via rewritings produced by our algorithms is feasible in practice.

7.4 GTGDs With Relations of Higher Arity

Finally, we computed the rewriting of GTGDs obtained by blowing up relation arity as described in Subsection 7.1 using a blowup factor of five. We did not use KAON2 since this system supports relations of arity at most two. Figure 5 summarizes our results. Out of 428 inputs, 187 were processed within the ten minute limit by our three systems, and 128 inputs were not processed by any system.

While HypDR performed best on GTGDs derived from ontologies, Figure 5 shows it to be worst-performing on higher-arity GTGDs: it successfully processed only 199 inputs within the ten minute timeout, whereas SkDR and ExbDR processed 238 and 274 inputs, respectively. This is mainly due to additional body atoms introduced by our "blowup" method: these increase the number of rules participating in an application of the HypDR inference rule, which makes selecting the participating rules harder.

This experiment proved to be more challenging, as most problems discussed in Section 6 became harder. For example, in ExbDR, higher arity of atoms increases the likelihood that an atom retrieved through a unification index does not unify with a given atom, and that the atoms of the selected GTGDs cannot be successfully matched. Subsumption indexing is also more difficult for similar reasons. However, the inputs used in this experiment consist of a large numbers of GTGDs with relations of arity ten, so they can be seen as a kind of a "stress test". Our algorithms were able to
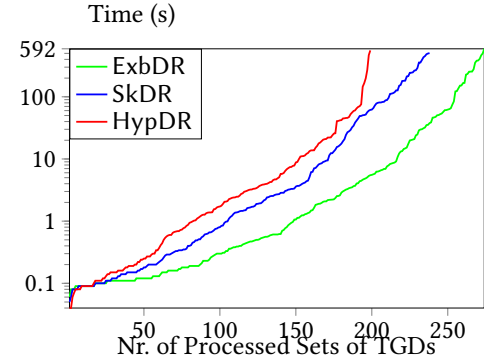
	ExbDR	SkDR	HypDR
# of Processed Inputs	274	238	199
Max. Processed Input Size	69,046	182,569	38,362
Max. Output Size	58,749	171,832	38,335
Max. Size Blowup	9.00	5.84	5.84
# Blowup $\geq$ 1.5	26	5	3
Time (s) Min.	0.06	0.05	0.04
Time (s) Max.	591.82	504.49	557.75
Time (s) Avg.	26.70	38.39	17.05
Time (s) Med.	0.61	1.65	1.72

	$time(Y)/time(X) \geq 10$			X and Y both fail		
X \ Y	ExbDR	SkDR	HypDR	ExbDR	SkDR	HypDR
ExbDR		61	87	154		
SkDR	11		21	128	190	
HypDR	6	4		148	184	229

Figure 5: Results for TGDs with Higher-Arity Relations

process more than half of such inputs, which leads us to believe that they can also handle more well-behaved GTGDs used in practice.

8 CONCLUSION

We presented several algorithms for rewriting a finite set of guarded TGDs into a Datalog program that entails the same base facts on each base instance. Our algorithms are based on a new framework that establishes a close connection between Datalog rewritings and a particular style of the chase. In future, we plan to generalize our framework to wider classes of TGDs, such as frontier-guarded TGDs, as well as provide rewritings for conjunctive queries under certain answer semantics. Moreover, we shall investigate whether the extension of our framework to disjunctive guarded TGDs [31] can be used to obtain practical algorithms for rewriting disjunctive guarded TGDs into disjunctive Datalog programs.

REFERENCES

[1] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus. 2018. Rewriting Guarded Existential Rules into Small Datalog Programs. In *ICDT*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 4:1–4:24.

[2] Günes Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. 2014. Diversified Stress Testing of RDF Data Management Systems. In *ISWC*. Springer, 197–212.

[3] Mario Alviano, Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. 2012. Magic-Sets for Datalog with Existential Quantifiers. In *Datalog 2.0*. Springer, 31–43.

[4] Antoine Amarilli and Michael Benedikt. 2022. When Can We Answer Queries Using Result-Bounded Data Interfaces? *Log. Methods Comput. Sci.* 18, 2 (2022), 14:1–-14:81.

[5] Hajnal Andréka, Johan van Benthem, and István Németi. 1998. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic* 27 (1998), 217–274.

[6] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider (Eds.). 2007. *The Description Logic Handbook: Theory, Implementation and Applications* (2nd ed.). Cambridge University Press, Cambridge, UK.

[7] M. Baaz, U. Egly, and A. Leitsch. 2001. Normal Form Transformations. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov (Eds.). Vol. I. Elsevier Science, Amsterdam, The Netherlands, Chapter 5, 273–333.

[8] Leo Bachmair and Harald Ganzinger. 2001. Resolution Theorem Proving. *Handbook of automated reasoning* 1 (2001), 19–99.

[9] J.-F. Baget, M. Leclère, M.-L. Mugnier, S. Rocher, and C. Sipieter. 2015. Graal: A Toolkit for Query Answering with Existential Rules. In *RuleML*. Springer, 328–344.

[10] Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. 2011. Walking the complexity lines for generalized guarded existential rules. In *IJCAI*. AAAI Press, 712–717.

[11] Vince Bárány, Michael Benedikt, and Balder Ten Cate. 2013. Rewriting Guarded Negation Queries. In *MFCS*. Springer, 98–110.

[12] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. 2018. The Vadalog System: Datalog-based Reasoning for Knowledge Graphs. *Proc. VLDB Endow.* 11, 9 (2018), 975–987.

[13] Michael Benedikt, Maxime Buron, Stefano Germano, Kevin Kappelmann, and Boris Motik. 2021. *Guarded Saturation*. GitHub. Retrieved July 4, 2022 from https://krr-oxford.github.io/Guarded-saturation/

[14] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the Chase. In *PODS*. ACM, 37–52.

[15] Andrea Calì, Georg Gottlob, and Michael Kifer. 2013. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *Journal of Artificial Intelligence Research* 48 (2013), 115–174.

[16] Andrea Calì, Domenico Lembo, and Riccardo Rosati. 2003. Query rewriting and answering under constraints in data integration systems. In *IJCAI*. Morgan Kaufmann, 16–21.

[17] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. 2017. Ontop: Answering SPARQL queries over relational databases. *Semantic Web* 8, 3 (2017), 471–487.

[18] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2007. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *J. Autom. Reason.* 39, 3 (2007), 385–429.

[19] Hans de Nivelle. 1998. A Resolution Decision Procedure for the Guarded Fragment. In *CADE*. Springer, 191–204.

[20] A. Deutsch, L. Popa, and V. Tannen. 2006. Query reformulation with constraints. *SIGMOD Record* 35, 1 (2006), 65–73.

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

[22] Mohamed Gaha, Arnaud Zinflou, Christian Langheit, Alexandre Bouffard, Mathieu Viau, and Luc Vouligny. 2013. An Ontology-Based Reasoning Approach for Electric Power Utilities. In *RR*. Springer, 95–108.

[23] H. Ganzinger and H. de Nivelle. 1999. A Superposition Decision Procedure for the Guarded Fragment with Equality. In *Proc. of the 14th IEEE Symposium on Logic in Computer Science (LICS '99)*. IEEE Computer Society, 295–305.

[24] Georg Gottlob, Sebastian Rudolph, and Mantas Simkus. 2014. Expressiveness of Guarded Existential Rule Languages. In *PODS*. ACM, 27–38.

[25] Alon Halevy, Anand Rajaraman, and Joann Ordille. 2006. Data Integration: The Teenage Years. In *VLDB*. ACM, 9–16.

[26] Alon Y. Halevy. 2001. Answering queries using views: A survey. *VLDB Journal* 10, 4 (2001), 270–294.

[27] Colin Hirsch. 2002. *Guarded Logics: Algorithms and Bisimulation*. Ph.D. Dissertation. RWTH Aachen, Aachen, Germany. Retrieved July 4, 2022 from http://www.umbrialogic.com/hirsch-thesis.pdf

[28] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. 2004. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. In *KR*. AAAI Press, 152–162.

[29] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. 2007. Reasoning in description logics by a reduction to disjunctive datalog. *Journal of Automated Reasoning* 39, 3 (2007), 351–384.

[30] David S. Johnson and Anthony C. Klug. 1984. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *JCSS* 28, 1 (1984), 167–189.

[31] Kevin Kappelmann. 2019. Decision Procedures for Guarded Logics. *CoRR* abs/1911.03679 (2019), 92. Retrieved July 4, 2022 from http://arxiv.org/abs/1911.03679

[32] Deepak Kapur and Paliath Narendran. 1986. NP-Completeness of the Set Unification and Matching Problems. In *CADE*. Springer, 489–495.

[33] Alon Y. Levy. 2000. *Logic-Based Techniques in Data Integration*. Kluwer Academic Publishers, Norwell, MA, USA, 575–595.

[34] Thomas Lukasiewicz, Andrea Calì, and Georg Gottlob. 2012. A General Datalog-Based Framework for Tractable Query Answering over Ontologies. *Journal of Web Semantics* 14, 0 (2012), 57–83.

[35] Bruno Marnette. 2012. Resolution and Datalog Rewriting Under Value Invention and Equality Constraints. *CoRR* abs/1212.0254 (2012), 12.

[36] William McCune and Larry Wos. 1997. Otter—The CADE-13 Competition Incarnations. *Journal of Automated Reasoning* 18, 2 (1997), 211–220.

[37] M. Meier. 2014. The backchase revisited. *VLDB J.* 23, 3 (2014), 495–516.

[38] Boris Motik. 2006. *Reasoning in description logics using resolution and deductive databases*. Ph.D. Dissertation. Karlsruhe Institute of Technology, Karlsruhe, Germany. Retrieved July 4, 2022 from http://digbib.ubka.uni-karlsruhe.de/volltexte/1000003797

[39] Boris Motik. 2022. *The KAON2 System*. Karslruhe Institute of Technology. Retrieved July 4, 2022 from http://kaon2.semanticweb.org/

[40] Oxford KR group. 2021. *Oxford Ontology Library*. Oxford University. Retrieved July 4, 2022 from http://krr-nas.cs.ox.ac.uk/ontologies/

[41] Mike Paterson and Mark N. Wegman. 1978. Linear Unification. *J. Comput. Syst. Sci.* 16, 2 (1978), 158–167.

[42] John Alan Robinson. 1965. A machine-oriented logic based on the resolution principle. *JACM* 12, 1 (1965), 23–41.

[43] Iztok Savnik. 2013. Index Data Structure for Fast Subset and Superset Queries. In *CD-ARES*. Springer, 134–148.

[44] Stephan Schulz. 2013. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In *Automated Reasoning and Mathematics*. Springer, 45–67.

[45] Mark E. Stickel. 1989. *The Path-Indexing Method for Indexing Terms*. Technical Report. SRI.

[46] Oxford Semantic Technologies. 2022. *The RDFox System*. Oxford Semantic Technologies. Retrieved July 4, 2022 from https://www.oxfordsemantic.tech/

[47] Moshe Y. Vardi. 1997. Why Is Modal Logic so Robustly Decidable?. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 31. American Mathematical Society, 149–184.

[48] Roberto De Virgilio, Giorgio Orsi, Letizia Tanca, and Riccardo Torlone. 2012. NYAYA: A System Supporting the Uniform Management of Large Sets of Semantic Data. In *ICDE*. IEEE Computer Society, 1309–1312.

[49] Zhe Wang, Peng Xiao, Kewen Wang, Zhiqiang Zhuang, and Hai Wan. 2021. Query Rewriting for Existential Rules via Efficient Datalog Rewriting. In *IJCAI*. ijcai.org, 1933–1939.

[50] Sen Zheng and Renate A. Schmidt. 2020. Deciding the Loosely Guarded Fragment and Querying Its Horn Fragment Using Resolution. In *AAAI*. AAAI Press, 3080–3087.

A PROOFS FOR SECTION 4: ONE-PASS CHASE PROOFS

In Section 4 we introduced the notion of a one-pass chase proof, which allows us to establish a completeness criterion for saturations that is tied to the chase. We provide details of the proofs in this appendix.

A.1 Proof of Theorem 4.2: Existence of One-Pass Chase Proofs

THEOREM 4.2. *For each base instance I, each finite set of GTGDs Σ in head-normal form, and each base fact F such that $I, \Sigma \models F$, there exists a one-pass tree-like chase proof of F from I and Σ.*

Throughout this section, we fix an arbitrary base instance I and a finite set of GTGDs Σ. It is known that $I, \Sigma \models F$ if and only if there exists a tree-like chase proof of F from I and Σ. We next prove Theorem 4.2 by showing that each such proof can be transformed to a one-pass chase proof of F from I and Σ. This argument was developed jointly with Antoine Amarilli, and it is related to proofs by Amarilli and Benedikt [4] and Kappelmann [31]; however, note that Definition 4.1 imposes slightly stronger conditions on one-pass chase sequences than related definitions in those works.

Towards our goal, we first state two basic properties of tree-like chase sequences. The first claim is a variation of the well-known fact that any chase tree produced for GTGDs represents a tree decomposition [15]. The second claim captures the idea that, as the chase progresses, facts may be added within a vertex, but this will not produced new guarded sets of terms.

LEMMA A.1. *Let $T_0, \ldots, T_n$ be an arbitrary tree-like chase sequence for I and Σ.*

1. *For each $0 \leq i \leq n$, all vertices v_1 and v_2 in T_i, each set G of ground terms that is Σ-guarded by $T_i(v_1)$ and by $T_i(v_2)$, and each vertex v_3 on the unique path in T_i between v_1 and v_2, set G is Σ-guarded by $T_i(v_3)$.*

2. *For each $0 \leq i \leq n$, each vertex v in T_i, each set G of ground terms that is Σ-guarded by $T_i(v)$, and each $0 \leq j \leq i$ such that T_j contains v, set G is Σ-guarded by $T_j(v)$.*

PROOF OF CLAIM 1. The proof is by induction on i with $0 \leq i \leq n$. For $i = 0$, chase tree T_0 contains just one vertex so the claim holds trivially. Now assume that the property holds for some $0 \leq i < n$ and consider ways in which T_{i+1} can be derived from T_i. First, T_{i+1} can be obtained by applying a chase step to T_i at vertex v with some GTGD $\tau \in \Sigma$. Let v_1 be the recently updated vertex of T_{i+1}. Thus, v_1 is either v or a fresh child of v. Moreover, consider each fact $R(\vec{t})$ derived by the step, each set of ground terms $G \subseteq \vec{t}$, each vertex v_2 such that G is Σ-guarded by $T_{i+1}(v_2)$, and each vertex v_3 on the unique path in T_{i+1} between v_1 and v_2. If G contains a labeled null that is freshly introduced in T_{i+1}, the claim holds trivially because v_2 and v_3 are necessarily the same as v_1. Otherwise, τ is guarded, so $T_i(v)$ contains a fact $S(\vec{u})$ such that $G \subseteq \vec{u}$. But then, G is Σ-guarded by $T_i(v_3)$ by the induction assumption. Moreover, $T_i(v_3) \subseteq T_{i+1}(v_3)$ ensures that G is Σ-guarded by $T_{i+1}(v_3)$, as required. Second, T_{i+1} can be obtained by applying a propagation step to T_i, but then the property clearly holds. □

PROOF OF CLAIM 2. The proof is by induction on i with $0 \leq i \leq n$. The base case for $i = 0$ is trivial. For the induction step, assume that the property holds for some i. If T_{i+1} is obtained from T_i by a chase step with a non-full GTGD, then the claim clearly holds for T_{i+1} because the step introduces a fresh vertex that does not occur in any T_j with $0 \leq j \leq i$. Otherwise, T_{i+1} is obtained by extending some $T_i(v)$, so consider an arbitrary fact $F \in T_{i+1}(v) \setminus T_i(v)$. Clearly, F is Σ-guarded by $T_i(v)$: if the step involves a full GTGD, then a body atom of the GTGD is matched to a fact $F' \in T_i(v)$ such that F is Σ-guarded by F'; moreover, if the step involves propagation, then by definition there exists a fact $F' \in T_i(v)$ such that F is Σ-guarded by F'. Thus, each set of ground terms G that is Σ-guarded by $T_{i+1}(v)$ is also Σ-guarded by $F' \in T_i(v)$, so the claim holds. □

In the rest of the proof, we show how to convert an arbitrary tree-like chase proof into a one-pass one through a series of transformations. Before proceeding, we next describe formally the types of chase sequence that we consider in our transformations.

Definition A.2.
- *A chase sequence is* local *if each propagation step in the sequence copies just one fact to either the parent or a child vertex.*
- *A chase sequence is* rootward *if each propagation step in the sequence copies just one fact from a child to its parent.*
- *A chase sequence is* almost one-pass *if it is rootward and each chase or propagation step is applied to the recently updated vertex or an ancestor thereof, and a chase step is applied only if a propagation step is not applicable to the recently updated vertex or an ancestor thereof.*

Note that facts can still be copied from a parent to a child in a rootward chase sequence, but this can be done only in chase steps with non-full GTGDs that introduce a child. Furthermore, the use of "almost" in the "almost one-pass" reflects the caveat that, in an almost one-pass chase sequence, a step can be applied to an ancestor of the recently updated vertex, thus "jumping rootward" in the tree, whereas such steps are forbidden in a one-pass chase sequence.

We capture formally the relationship between the chase sequences produced by our transformations using the notion introduced in Definition A.3.

Definition A.3. *A chase tree T is a* subset *of a chase tree T', written $T \subseteq T'$, if the tree of T is a subtree of T' (i.e., the root of T is the root of T', and whenever vertex v is a parent of vertex v' in T, then v is a parent of v' in T'), and $T(v) \subseteq T'(v)$ holds for each vertex v of T.*

We are now ready to present our transformations, which we capture in a series of lemmas. We next summarize the main intuitions.

- In Lemma A.4, we show that an arbitrary chase sequence can be transformed into a local chase sequence by "slowing down" propagation steps so that facts are copied only between vertices that are adjacent in a chase tree.

- In Lemma A.5, we show that each local chase sequence can be transformed into a rootward chase sequence. Intuitively, instead of propagating a fact from a parent to a child, we "regrow" a clone of the relevant child and the entire subtree underneath. The relevant fact is then copied as part of the chase step with the non-full GTGD that "regrows" the child's clone.

- In Lemma A.6, we show that each rootward chase sequence can be transformed to an almost one-pass chase sequence. The main difficulty arises due to the fact that steps in a rootward chase sequence can be applied to arbitrary vertices. We address this problem by shuffling and regrowing parts of the chase trees.

- Finally, in Lemma A.7, we show that each almost one-pass chase proof can be transformed to a one-pass chase proof by pruning irrelevant parts of the chase sequence.

LEMMA A.4. *For each tree-like chase sequence $T_0, \ldots, T_n$ for I and Σ, there exists a local tree-like chase sequence $\overline{T}_0, \ldots, \overline{T}_m$ for I and Σ such that $T_n \subseteq \overline{T}_m$.*

PROOF. Each propagation step in $T_0, \ldots, T_n$ that copies more than one fact can clearly be "expanded" into several steps, each copying just one fact. Moreover, due to Claim 1 of Lemma A.1, each propagation step that copies a fact F between vertices v and v' that are further apart can be "expanded" into several steps that propagate F to all vertices on the unique path between v and v'. □

LEMMA A.5. *For each local tree-like chase sequence $T_0, \ldots, T_n$ for I and Σ, there exists a rootward tree-like chase sequence $\overline{T}_0, \ldots, \overline{T}_m$ for I and Σ such that*

(S1) $T_n \subseteq \overline{T}_m$, and

(S2) for each vertex v in T_n that is introduced by a chase step with a non-full GTGD $\tau \in \Sigma$ and substitutions σ and σ', vertex v is introduced into some $\overline{T}_k$ with $0 \le k \le m$ by a chase step with the same τ, σ, and σ'.

PROOF. Let $T_0, \ldots, T_n$ be an arbitrary local tree-like chase sequence for I and Σ. We prove the claim by induction on $0 \le i \le n$. The induction base $i = 0$ holds trivially. For the induction step, we assume that the claim holds for some i with $0 \le i < n$. By the inductive assumption, there exists a rootward chase sequence $\overline{T}_0, \ldots, \overline{T}_j$ for I such that $T_i \subseteq \overline{T}_j$ and property (S2) holds. Let v be the vertex of T_i to which a chase or propagation step is applied to derive T_{i+1}. By Definition A.3, chase tree $\overline{T}_j$ contains vertex v and $T_i(v) \subseteq \overline{T}_j(v)$ holds. We now consider ways in which T_{i+1} can be derived from T_i.

Assume that T_{i+1} is obtained from T_i by a chase step with non-full TGD $\tau \in \Sigma$, and let v' be the child of v introduced by the step. Without loss of generality, we can choose v' and the fresh labeled nulls such that they do not occur in $\overline{T}_j$. Now let $\overline{T}_{j+1}$ be obtained from $\overline{T}_j$ by adding v' as a child of v and setting $\overline{T}_{j+1}(v') = T_{i+1}(v')$. Clearly, $\overline{T}_0, \ldots, \overline{T}_j, \overline{T}_{j+1}$ is a rootward chase sequence such that $T_{i+1} \subseteq \overline{T}_{j+1}$ and property (S2) hold, as required.

Assume that T_{i+1} is obtained from T_i by a chase step with a full TGD $\tau \in \Sigma$ deriving a fact F, or by a rootward propagation step that copies a fact F from $T_i(v)$ to the parent of v. Let v' be the recently updated vertex of T_{i+1}. Chase tree $\overline{T}_j$ clearly contains v'. If $F \in \overline{T}_j(v')$, then sequence $\overline{T}_0, \ldots, \overline{T}_j$ satisfies the inductive property. If T_{i+1} is obtained from T_i by a propagation step, then F is Σ-guarded by $T_i(v')$. But then, $T_i(v') \subseteq \overline{T}_j(v')$ ensures that F is also Σ-guarded by $\overline{T}_j(v')$ and thus the propagation step is applicable to vertices v and v' in $\overline{T}_j$. Now let $\overline{T}_{j+1}$ to be the same as $\overline{T}_j$ but with $\overline{T}_{j+1}(v') = \overline{T}_j(v') \cup \{F\}$ and with v' being the recently updated vertex. Clearly, $\overline{T}_0, \ldots, \overline{T}_j, \overline{T}_{j+1}$ is a rootward chase sequence satisfying $T_{i+1} \subseteq \overline{T}_{j+1}$, as required. Moreover, property (S2) holds by the induction hypothesis.

The only remaining case is when T_{i+1} is obtained from T_i by applying a propagation step that copies one fact F to a child v' of v. By Definition A.3, chase tree $\overline{T}_j$ contains vertex v' and $T_i(v') \subseteq \overline{T}_j(v')$ holds. Sequence $\overline{T}_0, \ldots, \overline{T}_j$ satisfies the inductive property if $F \in \overline{T}_j(v')$ holds, so we next assume $F \notin \overline{T}_j(v')$. We next show that we can simulate propagation by "replaying" the chase steps that generate v' and all of its descendants. Towards this goal, let T_k be the chase tree in the original sequence where v' is first introduced by applying a chase step with the non-full GTGD $\tau = \beta \to \exists \vec{y} \, \eta \in \Sigma$, and let σ and σ' be substitutions used in the step. By the inductive property (S2), there exists ℓ_0 with $0 < \ell_0 \le j$ such that v' is introduced in $\overline{T}_{\ell_0}$ as the result of applying a chase step with the same non-full TGD τ and substitutions σ and σ'. Finally, let $\overline{T}_{\ell_0}, \ldots, \overline{T}_{\ell_m}$ be the subsequence of $\overline{T}_0, \ldots, \overline{T}_j$ consisting of precisely those chase trees that were obtained by applying a chase or a propagation step to v' or a descendant of v'. In other words, the chase steps producing $\overline{T}_{\ell_0}, \ldots, \overline{T}_{\ell_m}$ are exactly the steps that we need to "replay" to simulate the propagation of F from v to v'.

Our objective is to "replay" the steps producing $\overline{T}_{\ell_0}, \ldots, \overline{T}_{\ell_m}$ so that they introduce exactly the same vertices and labeled nulls, which is needed because property (S1) talks about exact containment of the final chase trees of the two sequences (rather than containment up to isomorphism). A technical issue is that these vertices and labeled nulls already occur in the sequence $\overline{T}_0, \ldots, \overline{T}_j$; thus, if we extended this sequence directly, we could not "reapply" the chase steps with non-full GTGDs, which by definition introduce fresh vertices and labeled nulls. To get around this, we first perform the following renaming step. Let N be the set of labeled nulls introduced by the chase steps with non-full TGDs in subsequence $\overline{T}_{\ell_0}, \ldots, \overline{T}_{\ell_m}$, and let W be the set of introduced vertices (thus, W contains v' and all of its descendants). Moreover, let $\overline{U}_0, \ldots, \overline{U}_j$ be the chase sequence obtained by uniformly replacing in $\overline{T}_0, \ldots, \overline{T}_j$ each labeled null in N with a distinct, fresh labeled null, and by uniformly replacing each vertex $w \in W$ by a fresh vertex.

We next describe the chase trees that will be produced by "replaying" the steps producing the subsequence $\overline{T}_{\ell_0}, \ldots, \overline{T}_{\ell_m}$. Intuitively, we must "graft" the results of these steps onto $\overline{U}_j$: for v' or a descendant of v' we take the results of the chase steps in the subsequence, and for each other vertex we copy the content from $\overline{U}_j$. Formally, let $\overline{V}_0, \ldots, \overline{V}_m$ be the sequence obtained from the subsequence $\overline{T}_{\ell_0}, \ldots, \overline{T}_{\ell_m}$ using the following steps.

(R1) For each $0 \leq p \leq m$ and each vertex w in $\overline{U}_j$ such that w is neither v' nor a descendant of v' in $\overline{U}_j$, we set $\overline{V}_p(w) = \overline{U}_j(w)$.

(R2) For each $0 \leq p \leq m$ and each vertex w that occurs in $\overline{T}_{\ell_p}$ such that w is v' or a descendant of v' in $\overline{T}_{\ell_p}$, we set $\overline{V}_p(w) = \overline{T}_{\ell_p}(w)$.

(R3) We add to $\overline{V}_0(v')$ each fact $G \in \overline{U}_j(v)$ that is Σ-guarded by $\sigma'(\eta)$.

(R4) We analogously extend each $\overline{V}_p$ with $1 \leq p \leq m$ to ensure that each chase step with a non-full GTGD correctly propagates all relevant facts to a child.

We now argue that $\overline{U}_0, \ldots, \overline{U}_j, \overline{V}_0, \ldots, \overline{V}_m$ is a rootward chase sequence that satisfies properties (S1) and (S2). Towards this goal, we make the following observations.

- Sequence $\overline{U}_0, \ldots, \overline{U}_j$ is a rootward chase sequence produced by the same steps as $\overline{T}_0, \ldots, \overline{T}_j$, but with the vertices in W and labeled nulls in N uniformly renamed. Also, due to step (R4), $\overline{V}_0, \ldots, \overline{V}_m$ is a rootward chase sequence produced by the same steps as $\overline{T}_{\ell_0}, \ldots, \overline{T}_{\ell_m}$.

- Chase tree $\overline{V}_0$ coincides with $\overline{U}_j$ on each vertex that is not v' or a descendant of v'. Moreover, $\overline{U}_j$ does not contain a labeled null in N, and it does not contain v' or a descendant of v'; thus, $\overline{V}_0$ can be seen as the result of applying to $\overline{U}_j$ a chase step with the non-full GTGD τ and substitutions σ and σ' that introduces vertex v' as a child of v.

- We now show that property (S2) is satisfied—that is, that $T_{i+1} \subseteq \overline{V}_m$ holds. Towards this goal, consider an arbitrary vertex w occurring in T_{i+1}; by the induction assumption, we have $T_i(w) \subseteq \overline{T}_j(w)$. If w is neither v' nor a descendant thereof, then neither w nor a labeled null occurring in $T_i(w)$ was renamed in $\overline{U}_j$, so we have $\overline{T}_j(w) = \overline{U}_j(w) = \overline{V}_m(w)$, where the last equality is ensured by step (R1); thus, $T_{i+1}(w) = T_i(w) \subseteq \overline{V}_m(w)$ holds, as required. Now assume that w is v' or a descendant thereof. Then, $\overline{T}_j(w) = \overline{T}_{\ell_m}(w)$ holds by the fact that $\overline{T}_{\ell_m}$ is the last place in $\overline{T}_0, \ldots, \overline{T}_j$ where v' or a descendant of v' was modified, and $\overline{T}_{\ell_m}(w) = \overline{V}_m(w)$ holds by step (R2); putting it all together, we have $T_i(w) \subseteq \overline{V}_m(w)$. Now if w is not v' (i.e., w is a descendant of v'), then $T_{i+1}(w) = T_i(w) \subseteq \overline{V}_m(w)$ holds, as required. We finally consider the case when w is v', so $T_{i+1}(v') = T_i(v') \cup \{F\}$. Since the propagation step is applicable to T_i, fact F is Σ-guarded by $T_i(v')$. By Claim 2 of Lemma A.1, fact F is also Σ-guarded by $T_k(v')$. Finally, by the definition of a chase step with a non-full TGD, fact F is Σ-guarded by $\sigma'(\eta)$. But then, step (R3) ensures $F \in \overline{V}_0(v') \subseteq \overline{V}_m(v')$. Consequently, $T_{i+1}(v') \subseteq \overline{V}_m(v')$ holds, as required.

- We now show that property (S2) is satisfied. To this end, consider an arbitrary vertex w in T_{i+1} introduced by a chase step with a non-full GTGD τ and substitutions σ and σ'. If w is not v' or a descendant thereof, then the labeled nulls introduced by the chase step are not renamed in $\overline{U}_0, \ldots, \overline{U}_j$, so the claim holds by the induction assumption. Otherwise, the chase steps producing $\overline{V}_0, \ldots, \overline{V}_m$ are exactly the same as the chase steps producing $\overline{T}_{\ell_0}, \ldots, \overline{T}_{\ell_m}$, so the claim holds by the induction assumption too. □

LEMMA A.6. *For each rootward tree-like chase sequence $T_0, \ldots, T_n$ for I and Σ, there exists an almost one-pass chase sequence $\overline{T}_0, \ldots, \overline{T}_m$ for I and Σ such that $T_n \subseteq \overline{T}_m$.*

PROOF. Let $T_0, \ldots, T_n$ be an arbitrary rootward tree-like chase sequence for I and Σ. The induction base $i = 0$ holds trivially. For the induction step, we assume that the claim holds for some i with $0 \leq i < n$. By the inductive assumption, there exists an almost one-pass chase sequence $\overline{T}_0, \ldots, \overline{T}_j$ for I and Σ such that $T_i \subseteq \overline{T}_j$ holds. Now assume that T_{i+1} is obtained by applying a chase or a propagation step to some vertex v of T_i, and let k be the maximal number such that $0 \leq k \leq j$ and v is recently updated in $\overline{T}_k$. Such k clearly exists since v occurs in $\overline{T}_j$, and $T_i(v) \subseteq \overline{T}_k(v)$ holds because k is maximal. We now consider ways in which T_{i+1} can be derived from T_i.

Assume that T_{i+1} is obtained from T_i by a chase step with non-full GTGD $\tau \in \Sigma$ and substitutions σ and σ', and let v' be the child of v introduced by the step. Without loss of generality, we can choose v' and the fresh labeled nulls such that they do not occur in $\overline{T}_j$. We shall now "move" this chase step so that it is performed immediately after $\overline{T}_k$. Towards this goal, we describe the chase trees that are obtained by this move. For each p with $k \leq p \leq j$, let $\overline{U}_p$ be the chase tree obtained from $\overline{T}_p$ by adding vertex v' and letting $\overline{U}_p(v') = T_{i+1}(v')$. We now argue that $\overline{T}_0, \ldots, \overline{T}_k, \overline{U}_k, \ldots, \overline{U}_j$ is an almost one-pass chase sequence satisfying the conditions of the lemma.

- Chase tree $\overline{U}_k$ can be seen as obtained from $\overline{T}_k$ by a chase step with τ and substitutions σ and σ'. Moreover, for each p with $k \leq p < j$, chase tree $\overline{U}_{p+1}$ is obtained from $\overline{U}_p$ in the same way as $\overline{T}_{p+1}$ is obtained from $\overline{T}_p$. Thus, all preconditions of all chase steps are satisfied.

- Chase tree $\overline{U}_k$ is obtained from $\overline{T}_k$ by applying the chase step to the recently updated vertex v of $\overline{T}_k$. Moreover, if $k < j$, then $\overline{T}_{k+1}$ is obtained from $\overline{T}_k$ by applying a step to v or an ancestor of v, and so $\overline{U}_{k+1}$ is obtained from $\overline{U}_k$ by applying a step to an ancestor of the recently updated vertex of $\overline{U}_k$. Thus, the sequence is almost one-pass.

- The construction clearly satisfies $T_{i+1} \subseteq \overline{U}_j$.

In the rest of this proof we consider the case when T_{i+1} is obtained from T_i by a chase step with a full GTGD $\tau \in \Sigma$ deriving a fact F, or by a propagation step that copies a fact F from $T_i(v)$ to the parent of v. Let v' be the recently updated vertex of T_{i+1}. Chase tree $\overline{T}_j$ clearly

contains v'. If $F \in \overline{T}_k(v')$, then sequence $\overline{T}_0, \ldots, \overline{T}_j$ satisfies the inductive property, so we next assume that $F \notin \overline{T}_k(v')$ holds. We shall now transform $\overline{T}_0, \ldots, \overline{T}_j$ so that this step is applied immediately after $\overline{T}_k$, and fact F is propagated towards the root as far as possible. Since this will move the recently updated vertex towards the root, we will then "reapply" all relevant steps from $\overline{T}_0, \ldots, \overline{T}_j$ to "regrow" the relevant part of the sequence. In each case, we specify the structure of the chase trees and discuss the steps that produce these trees.

Let $\overline{U}_0$ be obtained from $\overline{T}_k$ by adding F to $\overline{T}_k(v)$. We argue that $\overline{U}_0$ can be seen as being obtained from $\overline{T}_k$ by the same step that produces T_{i+1} from T_i.

- If T_{i+1} is obtained from T_i by a chase step with a full GTGD, then $F \notin \overline{T}_k(v')$ holds ensures that the same step is applicable to $\overline{T}_k(v')$ (where $v' = v$).

- If T_{i+1} is obtained from T_i by a propagation step, then F is Σ-guarded by $T_i(v')$. But then, $T_i(v') \subseteq \overline{T}_j(v')$ ensures that F is also Σ-guarded by $\overline{T}_j(v')$, and Claim 2 of Lemma A.1 ensures that F is Σ-guarded by $\overline{T}_k(v')$. Thus, the propagation step is applicable to vertices v and v' in $\overline{T}_k$.

Moreover, let $\overline{U}_1, \ldots, \overline{U}_s$ be the chase trees obtained by propagating F starting from $\overline{U}_0$ towards the root using local steps as long as possible. Clearly, $\overline{T}_0, \ldots, \overline{T}_k, \overline{U}_0, \overline{U}_1, \ldots, \overline{U}_s$ is a correctly formed almost one-pass chase sequence. Let v'' be the recently updated vertex of $\overline{U}_s$,

We cannot simply append the step producing T_{k+1} after $\overline{U}_s$ because this step might not be applicable to v'' or an ancestor thereof. Thus, to obtain the chase sequence satisfying the claim of the lemma, we shall find a place in sequence $\overline{T}_0, \ldots, \overline{T}_j$ where vertex v'' is introduced, and we shall "replay" all steps from that point onwards. In doing so, we shall use chase steps that introduce the same vertices and labeled nulls, so we will first need to rename these in the sequence $\overline{T}_0, \ldots, \overline{T}_k, \overline{U}_0, \overline{U}_1, \ldots, \overline{U}_s$.

Let ℓ be the smallest integer such that $\overline{T}_\ell$ contains v''. Clearly, $\ell \leq k$ holds. Now let N be the set of labeled nulls introduced by applying a chase step to v'' or a descendant thereof, and let W the the set of descendants of v'' in the sequence $\overline{T}_0, \ldots, \overline{T}_j$. Moreover, let $\overline{T}'_0, \ldots, \overline{T}'_k, \overline{U}'_0, \overline{U}'_1, \ldots, \overline{U}'_s$ be the chase sequence obtained by uniformly replacing in $\overline{T}_0, \ldots, \overline{T}_k, \overline{U}_0, \overline{U}_1, \ldots, \overline{U}_s$ each labeled null in N with a distinct, fresh labeled null, and by uniformly replacing each vertex $w \in W$ by a fresh vertex.

We now transform chase trees $\overline{T}_{\ell+1}, \ldots, \overline{T}_j$ into chase tress $\overline{V}_{\ell+1}, \ldots, \overline{V}_j$ that reflect the result of "replaying" after $\overline{U}'_s$ the steps producing the former sequence. Intuitively, each $\overline{V}_p$ is a "union" of $\overline{U}_s$ and $\overline{T}_p$. Formally, for each p with $\ell < p \leq j$, we define $\overline{V}_p$ as follows.

(S1) The chase tree $\overline{V}_p$ contains the union of the vertices of $\overline{U}'_s$ and $\overline{T}_p$.

(S2) For each vertex w occurring only in $\overline{U}'_s$ (resp. $\overline{T}_p$), we define $\overline{V}_p(w) = \overline{U}'_s(w)$ (resp. $\overline{V}_p(w) = \overline{T}_p(w)$).

(S3) For each vertex w occurring in both $\overline{U}'_s$ and $\overline{T}_p$, we define $\overline{V}_p(w) = \overline{U}'_s(w) \cup \overline{T}_p(w)$.

(S4) If $\overline{T}_p$ is obtained by applying to a vertex w of $\overline{T}_{p-1}$ a chase step with a non-full GTGD $\tau = \forall \vec{x}[\beta \to \exists \vec{y}\, \eta]$ and substitutions σ and σ', then, for w' the child of w introduced by the step, we extend $\overline{V}_p(w')$ with each fact $G \in \overline{V}_{p-1}(w)$ that is Σ-guarded by $\sigma'(\eta)$.

We now argue that $\overline{T}'_0, \ldots, \overline{T}'_k, \overline{U}'_0, \overline{U}'_1, \ldots, \overline{U}'_s, \overline{V}_{\ell+1}, \ldots, \overline{V}_j$ contains an almost one-pass chase sequence for Σ and I that satisfies the conditions of this lemma.

- Sequence $\overline{T}'_0, \ldots, \overline{T}'_k, \overline{U}'_0, \overline{U}'_1, \ldots, \overline{U}'_s$ is clearly a valid almost one-pass chase sequence.

- For $\ell \leq p < j$, either $\overline{V}_{p+1}$ is obtained from $\overline{V}_p$ (or $\overline{U}'_s$ in case $p = \ell$) by the same step that produces $\overline{T}_{p+1}$ from $\overline{T}_p$, or the step is not applicable. In the latter case, we can simply drop such $\overline{V}_{p+1}$ from the sequence. By dropping all such $\overline{V}_{p+1}$, we clearly obtain a valid almost one-pass chase sequence.

- We have $T_i \subseteq \overline{T}_j$ by the induction assumption, and steps (S1)–(S3) clearly ensure $T_i \subseteq \overline{V}_j$. Moreover, T_{i+1} differs from T_i only in vertex v', where $T_{i+1}(v') = T(v') \cup \{F\}$ holds. Our construction, however, clearly ensures $F \in \overline{U}'_s(v'')$, and step (S4) ensures that F is propagated in each chase step with a non-full GTGD introducing a vertex on the unique path from v'' to v'. Thus, $T_{i+1} \subseteq \overline{V}_j$ holds. □

LEMMA A.7. *For each base fact F and each almost one-pass tree-like chase proof of F from I and Σ, there exists a one-pass tree-like chase proof of F from I and Σ.*

PROOF. Consider an arbitrary base fact F and an arbitrary almost one-pass tree-like chase proof $T_0, \ldots, T_n$ of F from I and Σ. Since F is a base fact, without loss of generality we can assume that F occurs in the facts of the root vertex. Now let T_i be the first chase tree that contains F in the root, and let W be the set containing each non-root vertex v occurring in any of the chase trees such that no propagation step is applied to v. We transform this proof to a one-pass proof as follows. First, we delete each T_j with $i < j \leq n$. Next, we delete in each remaining T_j each vertex $v \in W$ and each descendant of v. Finally, we delete each remaining T_j that is equal to T_{j+1}. After this transformation, every vertex has a propagation step applied to it. It is straightforward to see that the result is a one-pass tree-like chase sequence. Moreover, since F occurs in the root, the sequence is a tree-like chase proof of F from I and Σ. □

A.2 Proof of Proposition 4.7: Rewriting Criterion Using One-pass Chase Proofs

PROPOSITION 4.7. *A Datalog program Σ' is a rewriting of a finite set of GTGDs Σ in head-normal form if*

- *Σ' is a logical consequence of Σ,*

- *each Datalog rule of Σ is a logical consequence of Σ', and*
- *for each base instance I, each one-pass tree-like chase sequence $T_0, \ldots, T_n$ for I and Σ, and each loop $T_i, \ldots, T_j$ at the root vertex r with output fact F, there exist a Datalog rule $\beta \rightarrow H \in \Sigma'$ and a substitution σ such that $\sigma(\beta) \subseteq T_i(r)$ and $\sigma(H) = F$.*

PROOF. Let Σ and Σ' be as specified in the proposition, let I be an arbitrary base instance, and let F be an arbitrary base fact. Since Σ' is a logical consequence of Σ, it is clear that $I, \Sigma' \models F$ implies $I, \Sigma \models F$. Thus, we assume that $I, \Sigma \models F$ holds, and we prove that $I, \Sigma' \models F$ holds as well. By Theorem 4.2, there exists a one-pass tree-like chase proof $T_0, \ldots, T_n$ of F from I and Σ. Without loss of generality, we can assume that F is produced in the last step of the proof, and so the recently updated vertex of T_n is root vertex r. Let $i_0 < \cdots < i_m$ be exactly the indexes between 0 and n such that the recently updated vertex of T_{i_j} is r. We next construct a tree-like chase sequence $\overline{T}_0, \ldots, \overline{T}_k$ for I and Σ' such that $T_n(r) \subseteq \overline{T}_k(r)$. To formalize our inductive construction of this chase sequence, we shall also construct a sequence of indexes $\ell_0, \ldots, \ell_m$ such that $\ell_m = k$ and, for each j with $0 \leq j \leq m$, we have $T_{i_j}(r) \subseteq \overline{T}_{\ell_j}(r)$; in other words, each index ℓ_j helps us establish the inductive property by relating T_{i_j} and $\overline{T}_{\ell_j}$. For the base case, $i_0 = 0$ holds by the definition of a tree-like chase proof; thus, we set $\overline{T}_0 = T_0$ and $\ell_0 = 0$, and the required property clearly holds. For the inductive step, we consider arbitrary $0 < j \leq m$ such that the claim holds for $j - 1$, and assume that the sequence constructed thus far is $\overline{T}_{\ell_0}, \ldots, \overline{T}_{\ell_{j-1}}$. We have the following two cases.

- The recently updated vertex of $T_{i_j - 1}$ is r. Thus, $i_j - 1 = i_{j-1}$, and T_{i_j} is obtained from $T_{i_{j-1}}$ by a chase step with a full GTGD $\beta \rightarrow H \in \Sigma$ producing a fact $G \in T_{i_j}(r)$. The second condition of the proposition ensures that $\beta \rightarrow H$ is a logical consequence of Σ', so G can be derived from $\overline{T}_{\ell_1}(r)$ and the Datalog rules of Σ' using $\wp$ steps. We then define $\ell_j = \ell_{j-1} + \wp$, and we append the corresponding steps to obtain the sequence $\overline{T}_0, \ldots, \overline{T}_{\ell_{j-1}}, \ldots, \overline{T}_{\ell_j}$.
- Otherwise, $T_{i_{j-1}}, \ldots, T_{i_j}$ is a loop at the root vertex r with some output fact $G \in T_{i_j}(r)$. The third condition of the proposition ensures that there exists a Datalog rule $\beta \rightarrow H \in \Sigma'$ and a substitution σ such that $\sigma(\beta) \subseteq T_i(r)$ and $\sigma(H) = G$. We define $\ell_j = \ell_{j-1} + 1$, and we define $\overline{T}_{\ell_j}$ as the the chase tree containing just the root vertex r such that $\overline{T}_{\ell_j}(r) = \overline{T}_{\ell_{j-1}}(r) \cup \{G\}$; thus, $\overline{T}_{\ell_j}$ is obtained from $\overline{T}_{\ell_{j-1}}$ by applying the Datalog rule $\beta \rightarrow H \in \Sigma'$ to the root vertex r. Moreover, $T_{i_j}(r) \subseteq \overline{T}_{\ell_j}(r)$ clearly holds, as required. □

A.3 Proof of Proposition A.8: Properties of One-Pass Chase Proofs

We finally prove a property that will be needed in the proofs in Appendices B–E. This property intuitively ensures that, as soon as a fact F is derived in the child vertex of a loop such that F does not contain any null values introduced by the child, the loop is completed and F is propagated to the parent vertex.

PROPOSITION A.8. *For each loop $T_i, \ldots, T_j$ at a vertex v in a one-pass tree-like chase proof for some I and Σ, for $i < k \leq j$, and for v' the vertex introduced in T_{i+1}, set $\mathrm{terms}(T_k(v')) \setminus \mathrm{nulls}(T_{i+1}(v'))$ is Σ-guarded by $T_i(v)$.*

PROOF. Consider an arbitrary loop $T_i, \ldots, T_j$ at a vertex v in a one-pass tree-like chase proof for some I and Σ, and let v' be the child of v introduced by the chase step producing T_{i+1}. We prove the claim by induction on k with $i < k \leq j$. For the induction base $k = i + 1$, the definition of a chase step with a non-full GTGD clearly ensures this claim for $T_{i+1}(v')$. For the induction step, consider an arbitrary k such that the claim holds. Our claim holds trivially if $T_{k+1}(v') = T_k(v')$, so we assume that $T_{k+1}(v') \setminus T_k(v')$ contains exactly one fact F, which can be derived in one of the following two ways.

- Assume F is obtained by a propagation step to vertex v'. Then, F is Σ-guarded by $T_k(v')$, so $\mathrm{terms}(F) \subseteq \mathrm{terms}(T_k(v')) \cup \mathrm{consts}(\Sigma)$ holds.
- Assume F is obtained by applying a full GTGD $\tau \in \Sigma$ to $T_k(v')$ using a substitution σ. Then, τ contains a guard atom A in the body such that $\sigma(A) \subseteq T_k(v')$; moreover, the head τ contains all variables of A, and so we have $\mathrm{terms}(F) \subseteq \mathrm{terms}(T_k(v')) \cup \mathrm{consts}(\Sigma)$.

Either way, we have $\mathrm{terms}(T_{k+1}(v')) \subseteq \mathrm{terms}(T_k(v')) \cup \mathrm{consts}(\Sigma)$. By the induction assumption, set $\mathrm{terms}(T_k(v')) \setminus \mathrm{nulls}(T_{i+1}(v'))$ is Σ-guarded by $T_i(v)$, and so set $\mathrm{terms}(T_{k_1}(v')) \setminus \mathrm{nulls}(T_{i+1}(v'))$ is also Σ-guarded by $T_i(v)$, as required. □

B PROOFS FOR ExbDR

B.1 Proof of Proposition 5.7: Properties of ExbDR

PROPOSITION 5.7. *Each application of the ExbDR inference rule to τ, τ', and θ as in Definition 5.5 satisfies the following properties.*

1. *Some atom A'_i with $1 \leq i \leq n$ is a guard in τ'.*
2. *For each $1 \leq i \leq n$ such that A'_i is a guard of τ', and for σ the $\vec{y}$-MGU of A'_i and the corresponding atom A_i such that $\sigma(\vec{x}) \cap \vec{y} = \emptyset$, it is the case that $\mathrm{vars}(\sigma(A'_j)) \cap \vec{y} \neq \emptyset$ for each $1 \leq j \leq n$.*
3. *The result is a GTGD whose body and head width are at most $\mathrm{bwidth}(\Sigma)$ and $\mathrm{hwidth}(\Sigma)$, respectively.*

PROOF OF CLAIM 1. Let G be a guard for τ'. For the sake of a contradiction, assume that G is not one of the atoms $A'_1, \ldots, A'_n$—that is, $G \in \beta'$. Since $n \geq 1$, atom A'_1 in the body of τ' is matched to A_1 in the head of τ. Since τ is in head-normal form, A_1 contains at least one variable $y \in \vec{y}$. Moreover, the conditions of the ExbDR inference rule ensure $\theta(y) = y$. Since y does not occur in A'_1 and θ unifies A'_1 and

A_1, atom A'_1 contains at some position a variable z such that $\theta(z) = y$. Since G is a guard for τ', variable z occurs in G. Therefore, we have $\text{vars}(\theta(G)) \cap \vec{y} \neq \emptyset$, which contradicts the requirement $\text{vars}(\theta(\beta')) \cap \vec{y} = \emptyset$ of the ExbDR inference rule. $\qquad\square$

PROOF OF CLAIM 2. Consider arbitrary i such that $1 \leq i \leq n$ and A'_i is a guard of τ', and let σ be an MGU of A'_i and the corresponding atom A_i of τ. Since θ is a unifier of A'_i and A_i as well as of other pairs of atoms, there clearly exists a substitution ρ such that $\theta = \rho \circ \sigma$. Now consider an arbitrary A'_j with $1 \leq j \leq n$ in τ'. Substitution θ matches A'_j to the corresponding atom A_j in the head of τ. Since TGD τ is in head-normal form, atom A_j contains at least one variable $y \in \vec{y}$. Since $\theta(y) = y$, we necessarily have $y \in \text{vars}(\theta(A'_j))$. Consequently, atom A'_j contains some variable z such that $\theta(z) = y$. Since A'_i is a guard for τ', variable z occurs in A'_i. Now assume for the sake of a contradiction that $\sigma(z) \neq y$. Then $\sigma(z) = \sigma(x)$ for some $x \in \text{vars}(A_i)$ and $y = \theta(z) = \rho(\sigma(z)) = \rho(\sigma(x)) = \theta(x)$. However, this contradicts the requirement $\theta(\vec{x}) \cap \vec{y} = \emptyset$ of the ExbDR inference rule. $\qquad\square$

PROOF OF CLAIM 3. By Claim 1, there exists i with $1 \leq i \leq n$ such that atom A'_i is a guard for τ'. Thus, $\text{vars}(\beta') \cup \text{vars}(H') \subseteq \text{vars}(A'_i)$. The ExbDR inference rule ensures $\text{vars}(\theta(\beta')) \cap \vec{y} = \emptyset$, which in turn ensures $\text{vars}(\theta(\beta')) \subseteq \text{vars}(\theta(A'_i)) \setminus \vec{y}$. Now let G be a guard for τ. We clearly have $\text{vars}(\theta(\beta)) \subseteq \text{vars}(\theta(G))$. Moreover, $\theta(y_i) = y_i$ and $\theta(x) \cap \vec{y} = \emptyset$ ensure $\text{vars}(\theta(\eta)) \cup \text{vars}(\theta(A_i)) \subseteq \text{vars}(\theta(G)) \cup \vec{y}$. Thus, $\theta(G)$ is a guard for the TGD produced by the ExbDR inference rule. Finally, since G contains all variables of τ, the widths of the resulting TGD and τ are equal. $\qquad\square$

B.2 Proof of Theorem 5.8: Correctness and Complexity of ExbDR

THEOREM 5.8. *Program* $\text{ExbDR}(\Sigma)$ *is a Datalog rewriting of a finite set of GTGDs* Σ. *Moreover, the rewriting can be computed in time* $O(b^{r^d \cdot (w_b + c)^{da}} \cdot r^{d \cdot (w_h + c)^{da}})$ *for* r *the number of relations in* Σ, a *the maximum relation arity in* Σ, $w_b = \text{bwidth}(\Sigma)$, $w_h = \text{hwidth}(\Sigma)$, $c = |\text{consts}(\Sigma)|$, *and some* b *and* d.

PROOF OF CORRECTNESS. Let Σ' be the set Σ closed under the ExbDR inferences rule as specified in Definition 5.3. It is straightforward to see that Σ' is a logical consequence of Σ, so $\text{ExbDR}(\Sigma)$ is also a logical consequence of Σ. Moreover, $\text{ExbDR}(\Sigma)$ contains each full GTGD of Σ up to redundancy, so each full GTGD of Σ is logically entailed by $\text{ExbDR}(\Sigma)$. We next consider an arbitrary base instance I and a one-pass tree-like chase sequence for I and Σ, and we show the following property:

(♦) for each loop $T_i, \ldots, T_j$ in the sequence at some vertex v with output fact F, there exist a full GTGD $\beta \to H \in \Sigma'$ and a substitution σ such that $\sigma(\beta) \subseteq T_i(v)$ and $F = \sigma(H)$.

Since $\text{ExbDR}(\Sigma)$ contains all full TGDs of Σ' and this property holds for the root vertex r, Proposition 4.7 ensures that $\text{ExbDR}(\Sigma)$ is a rewriting of Σ.

Our proof is by induction on the length of the loop. The base case and the inductive step have the same structure, so we consider them jointly. Thus, consider an arbitrary loop $T_i, T_{i+1}, \ldots, T_{j-1}, T_j$ at vertex v in the sequence, and assume that the claim holds for all shorter loops in the sequence. By the definition of the loop, chase tree T_{i+1} is obtained from T_i by applying a chase step to some non-full TGD $\forall \vec{x}[\beta_0 \to \exists \vec{y} \, \eta_0] \in \Sigma$. Let σ_0 and σ'_0 be the substitutions used in this chase step, let $N = \text{nulls}(\text{rng}(\sigma'_0)) \setminus \text{nulls}(\text{rng}(\sigma_0))$, let v' be the child of v introduced in T_{i+1}, and let $S \subseteq T_i(v)$ be the facts that are copied to $T_{i+1}(v')$ because they are Σ-guarded by $\sigma'_0(\eta_0)$. Thus, we have $T_{i+1}(v') = S \cup \sigma'_0(\eta_0)$. By Proposition A.8 and the fact that a chase step is applied only if propagation to the parent is not applicable, the output fact of the loop is added to $T_{j-1}(v')$ in step $j-1$, and in T_j this fact is propagated back to $T_j(v)$. In other words, for each k with $i < k < j - 1$, each fact in $T_k(v') \setminus S$ contains at least one labeled null from N, or the fact would be Σ-guarded by $T_i(v)$ and thus propagated back to vertex v. We show that, in the loop $T_i, T_{i+1}, \ldots, T_{j-1}, T_j$ fixed above, the following property holds for each k with $i < k < j - 1$:

(◊) there exist a GTGD $\forall \vec{x}[\beta \to \exists \vec{y} \, \eta] \in \Sigma'$, a substitution σ such that $\sigma(\beta) \subseteq T_i(v)$, and a substitution σ' that extends σ by mapping $\vec{y}$ to fresh labeled nulls such that $T_k(v') \subseteq S \cup \sigma'(\eta)$.

We prove (◊) by induction on k. We have already proved the base case $k = i + 1$ above. For the inductive step, assume that (◊) holds for some k, so there exists a GTGD $\forall \vec{x}[\beta \to \exists \vec{y} \, \eta] \in \Sigma'$ and substitutions σ and σ' satisfying (◊) for k. Now consider T_{k+1}. Property (◊) holds by the inductive hypothesis if $T_{k+1}(v') = T_k(v')$—that is, if the step involves a descendant of v'. Otherwise, $T_{k+1}(v') = T_k(v') \cup \{G\}$ where fact G is obtained in one of the following two ways.

- A full GTGD in Σ derives G from $T_k(v')$. Set Σ' contains this GTGD up to redundancy, so by Definition 5.1 there exist a full GTGD $\beta'' \to H' \in \Sigma'$ and a substitution ρ such that $\rho(\beta'') \subseteq T_k(v')$ and $\rho(H') = G$.

- Fact G is the output of a loop at vertex v'. But then, this loop is shorter than $T_i, \ldots, T_j$ so, by property (♦), there exists a full GTGD $\beta'' \to H' \in \Sigma'$ and a substitution ρ such that $\rho(\beta'') \subseteq T_k(v')$ and $\rho(H') = G$.

Since η is in head-normal form, each atom in $\sigma'(\eta)$ contains at least one labeled null of N. Now let $A'_1, \ldots, A'_n$ be the atoms of β'' that are matched to the atoms in $\sigma'(\eta)$. Atom $\rho(H')$ contains at least one labeled null of N, so $n \geq 1$. Thus, we can assume that $\beta'' \to H'$ is of the form $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$ where $\{\rho(A'_1), \ldots, \rho(A'_n)\} \subseteq \sigma'(\eta)$ and $\rho(\beta') \subseteq S$. Also, since $\beta'' \to H'$ is guarded, at least one of A'_i is a guard for $\beta'' \to H'$. Let $A_1, \ldots, A_n$ be the atoms of η such that $\sigma'(A_i) = \rho(A'_i)$ for $1 \leq i \leq n$. Since σ' maps each $y \in \vec{y}$ to a distinct labeled null that does not occur in T_i, we have $\sigma'(\vec{x}) \cap \sigma'(\vec{y}) = \emptyset$. Thus, there exists a $\vec{y}$-MGU θ of $A_1, \ldots, A_n$ and $A'_1, \ldots, A'_n$ satisfying $\theta(\vec{x}) \cap \vec{y} = \emptyset$. Conjunction $\rho(\beta')$ does not contain a labeled null of N, so $\text{vars}(\theta(\beta')) \cap \vec{y} = \emptyset$ holds. Thus, the preconditions of the ExbDR inference rule are satisfied for $\forall \vec{x}[\beta \to \exists \vec{y} \, \eta]$ and $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$, so the ExbDR rule derives $\tau = \theta(\beta) \wedge \theta(\beta') \to \exists \vec{y} \, \theta(\eta) \wedge \theta(H')$.

Moreover, some A'_i is a guard so all variables of $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$ participate in unification, and thus we can extend σ and σ' to substitutions ζ and ζ', respectively, covering these variables such that $\zeta(\theta(\beta)) \cup \zeta(\theta(\beta')) \subseteq T_i(v)$ and $T_{k+1}(v') \subseteq S \cup \zeta'(\theta(\eta)) \cup \zeta'(\theta(H'))$. Set Σ' contains τ up to redundancy. Since $G \notin T_k(v')$, GTGD τ is not a syntactic tautology, so there exists a GTGD $\forall \vec{x}_1[\beta_1 \to \exists \vec{y}_1\, \eta_1] \in \Sigma'$ and substitution μ such that $\mathrm{dom}(\mu) = \vec{x}_1 \cup \vec{y}_1$, $\mu(\vec{x}_1) \subseteq \vec{x}_2$, $\mu(\vec{y}_1) \subseteq \vec{y}_1 \cup \vec{y}_2$ and $\mu(y) \neq \mu(y')$ for distinct y and y' in $\vec{y}_1$, and $\mu(\beta_1) \subseteq \theta(\beta) \cup \theta(\beta')$ and $\mu(\eta_1) \supseteq \theta(\eta) \wedge \theta(H')$. Now let σ_1 be the substitution defined as $\sigma_1(x) = \zeta(\mu(x))$ on each $x \in \vec{x}$, and let σ'_1 be the extension of σ_1 to $\vec{y}_1$ such that $\sigma'_1(y) = \zeta'(\mu(y))$ for each $y \in \vec{y}$. Clearly, $\sigma_1(\beta_1) \subseteq T_k(v')$ and $T_{k+1}(v') \subseteq S \cup \sigma'_1(\eta_1)$ hold, so property ($\diamond$) is satisfied.

To complete the proof, consider now the derivation of T_{j-1}. By property ($\diamond$), there exists a GTGD $\forall \vec{x}[\beta \to \exists \vec{y}\, \eta] \in \Sigma'$ and substitutions σ and σ' such that $\sigma(\beta) \subseteq T_i(v)$ and $T_{j-2}(v') = S \cup \sigma'(\eta)$. Then, as above, Σ' contains a full TGD of the form $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$ that satisfies $\rho(A'_1) \cup \cdots \cup \rho(A'_n) \subseteq \sigma'(\eta)$ and $\rho(\beta') \subseteq S$ for some substitution ρ. A minor difference is that $\rho(H')$ does not contain a labeled null introduced by $\sigma'(\eta_0)$, so $n = 0$ is possible; however, in such a case, this TGD immediately satisfies property ($\blacklozenge$). Moreover, if $n > 0$, then $\beta \to \exists \vec{y}\, \eta$ can again be resolved with $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$ to produce

$$\theta(\beta) \wedge \theta(\beta') \to \exists \vec{y}\, \theta(\eta) \wedge \theta(H') \in \Sigma'$$

satisfying $\mathrm{vars}(\theta(H')) \cap \vec{y} = \emptyset$. This TGD is transformed into head-normal form by Definition 5.3, so $\forall \vec{x}[\theta(\beta) \wedge \theta(\beta') \to \theta(H')]$ is contained in Σ' up to redundancy. But then, Σ' contains a full GTGD that satisfies property ($\blacklozenge$) by the same argument as above. $\qquad\square$

PROOF OF COMPLEXITY. Fix Σ, r, w_b, w_h, c, and a as stated in the theorem. The number of different body atoms of arity a constructed using r relations, w_b variables, and c constants is clearly bounded by $\ell_b = r \cdot (w_b + c)^a$. Moreover, by the third claim of Proposition 5.7, the number of variables in the head of each TGD is bounded by w_h, so the number of head atoms is bounded by $\ell_h = r \cdot (w_h + c)^a$. The body (resp. head) of each GTGD corresponds to a subset of these atoms, so number of different GTGDs up to variable renaming is bounded by $\wp = 2^{\ell_b} \cdot 2^{\ell_h}$. Thus, the ExbDR inference rule needs to be applied to at most $\wp^2 = 2^{2(\ell_b + \ell_h)}$ pairs of GTGDs. For each such pair, one might need to consider each possible way to match the ℓ_b body atoms of τ' to ℓ_h head atoms of τ, and there are at most $(\ell_h)^{\ell_b} \leq 2^{\ell_b \cdot \ell_h}$ of these. Consequently, unifier θ may need to be computed at most $2^{2(\ell_b + \ell_h)} \cdot 2^{\ell_b \cdot \ell_h} \leq 2^{5 \cdot \ell_b \cdot \ell_h} = 32^{\ell_b \cdot \ell_h}$ times. To check whether TGD $\tau_1 = \forall \vec{x}_1[\beta_1 \to \vec{y}_1\, \eta_1]$ is subsumed by $\tau_2 = \forall \vec{x}_2[\beta_2 \to \vec{y}_2\, \eta_2]$, we can proceed as follows. First, we consider all possible ways to match an atom of β_1 to an atom of β_2; since both conjunctions contain at most ℓ_b atoms, there are at most $\ell_b^{\ell_b} \leq 2^{\ell_b^2}$ such matchings. Second, we analogously consider each of at most $2^{(\ell_h)^2}$ ways to match an atom of η_2 to an atom of η_1. Once all atoms have been matched, we try to find a substitution μ satisfying Definition 5.1 in linear time. Thus, a subsumption check for pairs of TGDs takes at most $2^{\ell_b^2} \cdot 2^{\ell_h^2}$ steps. Finally, unification of atoms requires time that is linear in a, and all other steps require linear time too. $\qquad\square$

C PROOFS FOR SkDR

C.1 Proof of Proposition 5.12: Properties of SkDR

We reuse results by de Nivelle [19] about unification of atoms in guarded rules. The *variable depth* [19, Definition 3] of an atom is defined as -1 if the atom is ground, or as the maximum number of nested function symbols that contain a variable of the atom. Moreover, an atom is *weakly covering* [19, Definition 6] if each nonground functional subterm of the atom contains all variables of the atom. Finally, de Nivelle [19, Theorem 1] says that, for θ an MGU of weakly covering atoms A and B, atom $C = \theta(A) = \theta(B)$ is also weakly covering, the variable depth of C is bounded by the variable depth of A and B, and the number of variables of C is bounded by the number of variables of A and B too.

PROPOSITION 5.12. *Each application of the* SkDR *inference rule to rules τ and τ' as in Definition 5.10 produces a guarded rule.*

PROOF. Consider arbitrary rules $\tau = \beta \to H$ and $\tau' = A' \wedge \beta' \to H'$ and an MGU θ of H and A' satisfying the preconditions of the SkDR inference rule. Atom H thus contains a Skolem symbol, and rule τ is guarded; consequently, atom H is weakly covering, it contains a term of the form $f(\vec{t})$ where $\vec{t}$ consists of constants and all variables of the rule, and the variable depth of H is at most one. The corresponding atom A' can be of the following two forms.

- Atom A' is Skolem-free. But then, A' contains all variables of τ', and it is clearly weakly covering. By de Nivelle [19, Theorem 1], atom $\theta(A')$ is weakly covering and has variable depth at most one; consequently, each atom in rule $\theta(A') \wedge \theta(\beta') \to \theta(H')$ is weakly covering and has variable depth at most one. Moreover, the variable depth of $\theta(H)$ is also at most one, which can be only if θ maps each variable in H to another variable or a constant. Thus, each atom in rule $\theta(\beta) \to \theta(H)$ is weakly covering and has variable depth at most one; moreover, $\theta(\beta)$ contains an atom that contains all variables of the rule. But then, rule $\theta(\beta) \wedge \theta(\beta') \to \theta(H')$ is guarded, as required.

- Atom A' contains a Skolem symbol. But then, A' is weakly covering by the definition of guarded rules, and its variable depth is at most one. By de Nivelle [19, Theorem 1], atom $\theta(H) = \theta(A')$ is weakly covering and has variable depth at most one, which can be the case only if θ maps all variables to other variables or constants. Consequently, rules $\theta(\beta) \to \theta(H)$ and $\theta(A') \wedge \theta(\beta') \to \theta(H')$ are both guarded. But then, rule $\theta(\beta) \wedge \theta(\beta') \to \theta(H')$ is guarded, as required. $\qquad\square$

C.2 Proof of Theorem 5.13: Correctness and Complexity of SkDR

THEOREM 5.13. *Program* SkDR(Σ) *is a Datalog rewriting of a finite set of GTGDs Σ. Moreover, the rewriting can be computed in time $O(b^{r^d \cdot (e + w_b + c)^{da}})$ for r the number of relations in Σ, a the maximum relation arity in Σ, e the number of existential quantifiers in Σ, $w_b = \mathrm{bwidth}(\Sigma)$, $c = |\mathrm{consts}(\Sigma)|$, and some b and d.*

Proof of Correctness. Let Σ be an arbitrary finite set of GTGDs, and let Σ' be the set of rules obtained from Σ as specified in Definition 5.3. It is straightforward to see that Σ' is a logical consequence of the Skolemization of Σ, so $\mathrm{SkDR}(\Sigma)$ is also a logical consequence of Σ. Moreover, $\mathrm{SkDR}(\Sigma)$ contains each full TGD of Σ up to redundancy, so each full TGD of Σ is logically entailed by $\mathrm{SkDR}(\Sigma)$. We next consider an arbitrary base instance I and a one-pass tree-like chase sequence for I and Σ, and we show the following property:

> ($\blacklozenge$) for each loop $T_i, \ldots, T_j$ in the sequence at some vertex v with output fact F, there exist a Skolem-free rule $\beta \to H \in \Sigma'$ and a substitution σ such that $\sigma(\beta) \subseteq T_i(v)$ and $F = \sigma(H)$.

Since $\mathrm{SkDR}(\Sigma)$ contains all Skolem-free rules of Σ' and this property holds for the root vertex r, Proposition 4.7 ensures that $\mathrm{SkDR}(\Sigma)$ is a rewriting of Σ.

Our proof is by induction in the length of the loop. The base case and the inductive step have the same structure, so we consider them jointly. Thus, consider an arbitrary loop $T_i, T_{i+1}, \ldots, T_{j-1}, T_j$ at vertex v in the sequence, and assume that the claim holds for all shorter loops in the sequence. By the definition of a loop, chase tree T_{i+1} is obtained from T_i by applying a chase step to some non-full GTGD $\tau \in \Sigma$ and substitution γ. Let v' be the child of v introduced in T_{i+1}, let $S \subseteq T_i(v)$ be the facts that are copied to $T_{i+1}(v')$ because they are Σ-guarded by the instantiated head of τ, let $N = \{n_1, \ldots, n_m\}$ be the set of labeled nulls introduced in the chase step for the existentially quantified variables $y_1, \ldots, y_m$ of τ, let ν be a function that maps each labeled null n_i to the ground term $f_i(\gamma(\vec{x}))$ where f_i is the symbol used in the Skolemization of y_i. For U a set of facts, let $\nu(U)$ be the result of replacing each occurrence of a labeled null $n \in \mathrm{dom}(\nu)$ in U with $\nu(n)$ and eliminating any duplicate facts in the result. Clearly, the inverse function ν^- is well-defined, and we define $\nu^-(U)$ for U a set of facts in the obvious way. By Proposition A.8 and the fact that propagation is applied eagerly, the output fact of the loop is added to $T_{j-1}(v')$ in step $j-1$, and in T_j this fact is propagated back to $T_j(v)$. In other words, for each k with $i < k < j - 1$, each fact in $T_k(v') \setminus S$ contains at least one labeled null from N, or the fact would be Σ-guarded by $T_i(v)$ and would thus be propagated back to vertex v. We now show that the following property holds for each k with $i < k \le j - 1$:

> ($\lozenge$) for each fact $G \in T_k(v') \setminus S$, there exist a rule $\beta \to H \in \Sigma'$ and a substitution σ such that β is Skolem-free, $\sigma(\beta) \subseteq \nu(T_i(v))$, and $\sigma(H) = \nu(G)$.

Property ($\lozenge$) implies ($\blacklozenge$): fact F does not contain a labeled null from N, so the rule $\beta \to H \in \Sigma'$ whose existence is implied by ($\lozenge$) for $k = j - 1$ is actually a Skolem-free rule that satisfies ($\blacklozenge$).

We next prove property ($\lozenge$) by a nested induction on k. For the base case $k = i + 1$, property ($\lozenge$) holds due to the fact that Σ' contains the rules obtained by Skolemizing GTGD τ. For the inductive step, assume that ($\lozenge$) holds for some k and consider the possible ways to obtain T_{k+1} from T_k. Property ($\lozenge$) holds by the inductive hypothesis if $T_{k+1}(v') = T_k(v')$—that is, if the step involves a descendant of v'. Otherwise, $T_{k+1}(v') = T_k(v') \cup \{G\}$ where fact G is obtained in one of the following two ways.

- A full TGD in Σ derives G from $T_k(v')$. Set Σ' contains this TGD up to redundancy, so by Definition 5.1 there exist a Skolem-free rule $\beta'' \to H' \in \Sigma'$ and a substitution σ' such that $\sigma'(\beta'') \subseteq \nu(T_k(v'))$ and $\sigma'(H') = \nu(G)$.
- Fact G is the output of a loop at vertex v'. But then, this loop is shorter than $T_i, \ldots, T_j$ so, by property ($\blacklozenge$), there exist a Skolem-free rule $\beta'' \to H' \in \Sigma'$ and a substitution σ' such that $\sigma'(\beta'') \subseteq \nu(T_k(v'))$ and $\sigma'(H') = \nu(G)$.

Now let $W = \{B' \in \beta'' \mid \sigma'(B') \notin S\}$. We next show that set Σ' contains up to redundancy the result of "resolving away" each atom $B' \in W$. A slight complication arises due to the fact that the SkDR inference rule considers only two rules at a time, and that the result of each inference is contained in Σ' up to redundancy. Thus, we will achieve our goal by showing that the SkDR inference rule can be applied up to $n = |W|$ times. Our proof is by induction on $1 \le \ell \le n$. Towards this goal, we shall define n rules $\beta''_\ell \to H'_\ell$, substitutions σ'_ℓ, and sets of atoms $W = W_0 \supsetneq \cdots \supsetneq W_n$ for ℓ with $0 \le \ell \le n$ satisfying the following invariant:

> ($*$) $\sigma'_\ell(\beta''_\ell) \subseteq S \cup \{\sigma'(B') \mid B' \in W_\ell\}$ and $\sigma'_\ell(H'_\ell) = \sigma'(H')$.

For $\ell = n$, we have $W_n = \emptyset$, and so property ($*$) implies property ($\lozenge$), as required. Our construction proceeds as follows.

For the base case $\ell = 0$, property ($*$) clearly holds for $\beta''_0 = \beta''$, $\sigma'_0 = \sigma'$, and let $W_0 = W$. For the induction step, assume that ($*$) holds for some $0 \le \ell < n$, so $\beta''_\ell \to H'_\ell$, σ'_ℓ, and W_ℓ satisfying ($*$) have been defined. First, assume that there exists $B' \in W_\ell$ such that $\sigma'(B') \notin \sigma'_\ell(\beta''_\ell)$. Then, property ($*$) clearly holds for $\beta''_{\ell+1} = \beta''_\ell$, $H'_{\ell+1} = H'_\ell$, $\sigma'_{\ell+1} = \sigma'_\ell$, and $W_{\ell+1} = W_\ell \setminus \{B'\}$. Otherwise, we consider the following possibilities.

- If rule $\beta''_\ell \to H'_\ell$ is Skolem-free, the rule is of the form $A'_\ell \wedge \beta'_\ell \to H'_\ell$ where A'_ℓ contains all variables of the rule.
- Otherwise, rule $\beta''_\ell \to H'_\ell$ is of the form $A'_\ell \wedge \beta'_\ell \to H'_\ell$ where atom A'_ℓ contains a Skolem symbol, in which case this atom contains all variables of the rule.

Either way, there exists $B' \in W_\ell$ such that $\sigma'_\ell(A'_\ell) = \sigma'(B')$ where $\sigma'(B') \in T_k(v') \setminus S$. Thus, by property ($\lozenge$), these exist a rule $\beta_\ell \to H_\ell \in \Sigma'$ and a substitution σ_ℓ such that β_ℓ is Skolem-free, $\sigma_\ell(\beta_\ell) \subseteq \nu(T_i(v))$, and $\sigma_\ell(H_\ell) = \sigma'(B')$; the last observation ensures that H_ℓ contains a Skolem symbol. Moreover, there exists an MGU θ_ℓ of H_ℓ and A'_ℓ, so the SkDR inference rule is applicable to $\beta_\ell \to H_\ell$ and $A'_\ell \wedge \beta'_\ell \to H'_\ell$, and Σ' contains rule $\theta_\ell(\beta_\ell) \wedge \theta_\ell(\beta'_\ell) \to \theta_\ell(H_\ell)$ up to redundancy. Now let $\zeta_\ell = (\sigma_\ell \cup \sigma'_\ell) \circ \theta$ be the composition of $\sigma_\ell \cup \sigma'_\ell$ and θ; note that substitution $\sigma_\ell \cup \sigma'_\ell$ is correctly defined because rules $\beta_\ell \to H_\ell$ and $A'_\ell \wedge \beta'_\ell \to H'_\ell$ do not share variables. Moreover, let $W_{\ell+1} = W_\ell \setminus \{B'\}$. We clearly have $\zeta_\ell(\theta_\ell(\beta_\ell)) \subseteq S, \zeta_\ell(\theta_\ell(\beta'_\ell)) \subseteq S \cup \{\sigma'(C') \mid C' \in W_{\ell+1}\}$, and $\zeta_\ell(\theta_\ell(H)) = \sigma'(H')$. Since $G \notin T_k(v')$, rule $\theta_\ell(\beta_\ell) \wedge \theta_\ell(\beta'_\ell) \to \theta_\ell(H_\ell)$ is not a syntactic tautology. Thus, by Definition 5.1, there exist a rule $\beta''_{\ell+1} \to H'_{\ell+1} \in \Sigma'$ and substitution $\mu_{\ell+1}$ such that $\mu_{\ell+1}(\beta''_{\ell+1}) \subseteq \theta_\ell(\beta_\ell) \cup \theta_\ell(\beta'_\ell)$ and $\mu_{\ell+1}(H'_{\ell+1}) = \theta_\ell(H'_\ell)$. Now let $\sigma'_{\ell+1}$ be the substitution defined on each variable x in $\beta''_{\ell+1} \to H'_{\ell+1}$ such that $\sigma'_{\ell+1}(x) = \zeta_\ell(\mu_{\ell+1}(x))$. Then, property ($*$) clearly holds for $\beta''_{\ell+1} \to H'_{\ell+1}$, $\sigma'_{\ell+1}$, and $W_{\ell+1}$, as required. $\qquad\square$

Proof of Complexity. Fix Σ, r, w_b, e, c, and a as stated in the theorem. Skolemizing a GTGD $\forall \vec{x}[\beta \rightarrow \exists \vec{y}\, \eta]$ produces guarded rules in which each atom is of the form $R(t_1, \ldots, t_n)$ such that each t_i is a constant, a variable from $\vec{x}$, or a term of the form $f(\vec{x})$ where f is a Skolem symbol. Moreover, each atom obtained from $R(t_1, \ldots, t_n)$ by the SkDR inference rule is obtained by replacing a variable in $\vec{x}$ with another variable or a constant. Thus, atom $R(t_1, \ldots, t_n)$ cannot contain more than $|\vec{x}|$ variables. Since the number of different symbols obtained by Skolemization is clearly bounded by e, the number of different atoms of such form is bounded by $\ell = r \cdot (w_b + e + c)^a$. The body of each guarded rule corresponds to a subset of these atoms, so the number of different rules up to variable remaining is bounded by $2^\ell \cdot \ell \leq 2^\ell \cdot 2^\ell = 2^{2\ell} = \wp$. By Definition 5.3, the result of applying the SkDR inference rule is retained in set Σ' only if the set does not contain a variable renaming of the result. Thus, the SkDR inference rule needs to be applied to at most $\wp^2 = 2^{4\ell}$ pairs of rules. For each pair, one might need to unify at most ℓ body atoms of one rule with the head atom of the other rule, so the unifier θ may need to be computed at most $\wp^2 \cdot \ell \leq \wp^2 \cdot 2^\ell = 2^{5\ell} = 32^\ell$ times. We can check subsumption between a pair of rules analogously to Theorem 5.8: for each of at most 2^{ℓ^2} ways to match the body atoms of one rule to the body atoms of another rule, we try to find a substitution μ satisfying Definition 5.1. Finally, unification of atoms requires time that is linear in a, and all other steps require linear time too. □

D PROOFS FOR HypDR

D.1 Proof of Proposition 5.18: Properties of HypDR

Proposition 5.18. *Each application of the HypDR inference rule to rules $\tau_1, \ldots, \tau_n$ and τ' as in Definition 5.16 produces a guarded rule.*

Proof. Consider arbitrary rules $\tau_i = \beta_i \rightarrow H_i$ with $1 \leq i \leq n$ such that β_i is Skolem-free and H_i contains a Skolem symbol, a Skolem-free rule $\tau' = A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H'$, and an MGU θ of $H_1, \ldots, H_n$ and $A'_1, \cdots A'_n$ satisfying the preconditions of the HypDR inference rule. Rule τ_1 contains a term with a Skolem symbol in the head, and this term is unified with a variable, say x, occurring in a Skolem-free body atom A'_1 of rule τ'. Moreover, rule τ' is guarded, so the body of the rule contains a Skolem-free atom G that contains all variables of the rule; thus, G also contains x. Since $\theta(x)$ contains a Skolem symbol, $\theta(G)$ contains a Skolem symbol too. However, $\theta(\beta')$ is Skolem-free, so G must be one of the atoms $A'_1, \ldots, A'_n$ from the body of rule τ' that are participating in the HypDR inference rule. But then, we can show that the result of the inference is guarded analogously to the proof of Proposition 5.12. □

D.2 Proof of Theorem 5.19: Correctness and Complexity of HypDR

Theorem 5.19. *Program HypDR(Σ) is a Datalog rewriting of a finite set of GTGDs Σ. Moreover, the rewriting can be computed in time time $O(b^{r^d \cdot (e+w_b+c)^{da}})$ for r the number of relations in Σ, a the maximum relation arity in Σ, e the number of existential quantifiers in Σ, $w_b = \mathsf{bwidth}(\Sigma)$, $c = |\mathsf{consts}(\Sigma)|$, and some b and d.*

Proof of Correctness. The correctness proof for HypDR is almost identical to the correctness proof in Theorem 5.13, so we outline just the difference. In particular, we wish to prove properties ($\blacklozenge$) and ($\lozenge$) exactly as stated in Theorem 5.13 using the same proof structure. In the proof of property ($\lozenge$), we establish existence of a Skolem-free rule $\beta'' \rightarrow H' \in \Sigma'$ and a substitution σ' such that $\sigma'(\beta'') \subseteq \nu(T_k(v'))$ and $\sigma'(H') = \nu(G)$ in exactly the same way. The difference to the proof of Theorem 5.13 is that we "resolve away" all relevant body atoms of β'' in one step. To this end, let $A'_1, \ldots, A'_n$ be precisely the atoms of β'' such that $\sigma'(A'_i) \notin S$ for each $1 \leq i \leq n$. Thus, we can assume that the rule is of the form $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H'$, and $\sigma'(\beta') \subseteq S$ clearly holds. By property ($\lozenge$), for each $1 \leq \ell \leq n$, there exist a rule $\beta_\ell \rightarrow H_\ell \in \Sigma'$ and substitution σ_ℓ such that β_ℓ is Skolem-free and $\sigma_\ell(H_\ell) = \sigma'(A'_\ell)$; the last observation ensures that H_ℓ contains a Skolem symbol. Finally, there exists an MGU θ of $H_1, \ldots, H_n$ and $A'_1, \ldots, A'_n$. Since $\sigma'(\beta') \subseteq S$, conjunction $\theta(\beta')$ is Skolem-free. Thus, the HypDR inference rule is applicable to $\beta_1 \rightarrow H_1, \ldots, \beta_n \rightarrow H_n$ and $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H'$, so set Σ' contains rule $\theta(\beta_1) \wedge \cdots \wedge \theta(\beta_n) \wedge \theta(\beta') \rightarrow \theta(H')$ up to redundancy. Since no premises share variables, substitution $\sigma_1 \cup \cdots \cup \sigma_n \cup \sigma'$ is correctly defined, so let ζ be the composition of $\sigma_1 \cup \cdots \cup \sigma_n \cup \sigma'$ and θ. Clearly, we have $\zeta(\theta(\beta_1)) \cup \cdots \cup \zeta(\theta(\beta_n)) \cup \zeta(\theta(\beta')) \subseteq S$ and $\zeta(\theta(H')) = \sigma'(H') = \nu(G)$. Since $G \notin T_k(v')$, rule $\theta(\beta_1) \wedge \cdots \wedge \theta(\beta_n) \wedge \theta(\beta') \rightarrow \theta(H')$ is not a syntactic tautology so, by Definition 5.1, there exist a rule $\beta \rightarrow H \in \Sigma'$ and substitution μ such that $\mu(\beta) \subseteq \theta(\beta_1) \cup \cdots \cup \theta(\beta_n) \cup \theta(\beta')$ and $\mu(H) = \theta(H')$. Let σ be the substitution defined on each variable x in $\beta \rightarrow H$ such that $\sigma(x) = \zeta(\mu(x))$. Then, $\sigma(\beta) \subseteq S$ and $\sigma(H) = \sigma'(H') = \nu(G)$, as required for property ($\lozenge$). □

Proof of Complexity. Fix Σ, r, w_b, e, c, and a as stated in the theorem. In the same way as in the complexity proof of Theorem 5.13, the number of different atoms can be bounded by $\ell = r \cdot (w_b + e + c)^a$, and the number of different rules can be bounded by $\wp = 2^{2\ell}$. Now we can apply the HypDR inference rule as follows: we choose one of the $\wp$ rules that plays the role of τ' and then, for each of the at most ℓ body atoms in τ', we select one of the $\wp$ rules that play the role of rules τ_i. Hence, there are at most $\wp \cdot \wp^\ell = \wp^{\ell+1}$ different applications of the HypDR inference rule. Thus, we may need to compute the unifier θ at most $(2^{2\ell})^{\ell+1} = 2^{2\ell^2 + 2\ell} \leq 2^{3\ell^2}$ times. Finally, the times needed for subsumption checking, unification, and all other steps can be bounded analogously as in the complexity proof of Theorem 5.13. □

E THE FullDR ALGORITHM: CREATING DATALOG RULES DIRECTLY

The algorithms presented in the body of the paper all create the Datalog rules needed for the final rewriting as well as intermediate non-full TGDs or rules that are discarded after all inferences are performed. We now present an algorithm that produces *only* Datalog rules. Similar algorithms have appeared in the prior literature [4]. After presenting such an algorithm, we explain the shortcomings of this approach.

Definition E.1. *The* Full Datalog Rewriting *inference rule* FullDR *can be applied in two ways, depending on the types of TGDs it takes.*

- *The* (COMPOSE) *variant of the* FullDR *inference rule takes full TGDs*

$$\tau = \forall \vec{x}[\beta \rightarrow A] \qquad and \qquad \tau' = \forall \vec{z}[A' \wedge \beta' \rightarrow H']$$

  and a substitution θ such that

  – $\theta(A) = \theta(A')$,

  – $\mathrm{dom}(\theta) = \vec{x} \cup \vec{z}$, *and*

  – $\mathrm{rng}(\theta) \subseteq \vec{w} \cup \mathrm{consts}(\tau) \cup \mathrm{consts}(\tau')$ *where $\vec{w}$ is a vector of* $\mathrm{hwidth}(\Sigma) + |\mathrm{consts}(\Sigma)|$ *variables different from* $\vec{x} \cup \vec{y} \cup \vec{z}$,

  and it derives

$$\theta(\beta) \wedge \theta(\beta') \rightarrow \theta(H').$$

- *The* (PROPAGATE) *variant of the* FullDR *inference rule takes TGDs*

$$\tau = \forall \vec{x}[\beta \rightarrow \exists \vec{y}\, \eta \wedge A_1 \wedge \cdots \wedge A_n] \qquad and \qquad \tau' = \forall \vec{z}[A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H']$$

  and a substitution θ such that

  – $\theta(A_i) = \theta(A'_i)$ *for each i with $1 \leq i \leq n$,*

  – $\mathrm{dom}(\theta) = \vec{x} \cup \vec{z}$,

  – $\mathrm{rng}(\theta) \subseteq \vec{w} \cup \vec{y} \cup \mathrm{consts}(\tau) \cup \mathrm{consts}(\tau')$ *where $\vec{w}$ is a vector of* $\mathrm{hwidth}(\Sigma) + |\mathrm{consts}(\Sigma)|$ *variables different from* $\vec{x} \cup \vec{y} \cup \vec{z}$,

  – $\theta(\vec{x}) \cap \vec{y} = \emptyset$, *and*

  – $\mathrm{vars}(\theta(\beta')) \cap \vec{y} = \emptyset$ *and* $\mathrm{vars}(\theta(H')) \cap \vec{y} = \emptyset$,

  and it derives

$$\theta(\beta) \wedge \theta(\beta') \rightarrow \theta(H').$$

THEOREM E.2. *Program* FullDR(Σ) *is a Datalog rewriting of a finite set of GTGDs Σ. Moreover, the rewriting can be computed in time $O(b^{r^d \cdot (w+c)^{da}})$ for r the number of relations in Σ, a the maximum relation arity in Σ, $w = \mathrm{width}(\Sigma)$, $c = |\mathrm{consts}(\Sigma)|$, and some b and d.*

PROOF OF CORRECTNESS. The proof follows the same structure as the correctness proof of Theorem 5.8: we show that property (♦) holds for each loop on a one-pass tree-like chase sequence for I and Σ; a minor difference is that the TGD whose existence is implied by (♦) is not necessarily guarded, but has width bounded by $\mathrm{hwidth}(\sigma)$. To this end, we consider an arbitrary loop $T_i, T_{i+1}, \ldots, T_{j-1}, T_j$ at vertex v in the sequence, and assume that the claim holds for all shorter loops in the sequence. By the definition of the loop, chase tree T_{i+1} is obtained from T_i by applying a chase step to some non-full TGD $\forall \vec{x}[\beta_0 \rightarrow \exists \vec{y}\, \eta_0] \in \Sigma$. Let σ_0 and σ'_0 be the substitutions used in this chase step, and let v' be the child of v introduced in T_{i+1}. Note that $T_{i+1}(v')$ contains at most $\mathrm{hwidth}(\Sigma) + |\mathrm{consts}(\Sigma)|$ distinct terms. We show by another induction on k that the following property holds for each k with $i < k \leq j - 1$:

(◊) for each fact $G \in T_k(v') \setminus T_{i+1}(v')$, there exist a full TGD $\forall \vec{x}[\beta \rightarrow H] \in \Sigma'$ of width at most $\mathrm{width}(\Sigma)$ and a substitution σ such that $\sigma(\beta) \subseteq T_{i+1}(v')$ and $\sigma(H) = G$.

For the base case $k = i + 1$, property (◊) holds vacuously because $T_k(v') \setminus T_{i+1}(v') = \emptyset$. For the inductive step, assume that (◊) holds for some k and consider the possible ways to obtain T_{k+1} from T_k. Property (◊) holds by the inductive hypothesis if $T_{k+1}(v') = T_k(v')$—that is, if the step involves a descendant of v'. Otherwise, $T_{k+1}(v') = T_k(v') \cup \{G\}$ where fact G is obtained in one of the following two ways.

- A full TGD in Σ derives G from $T_k(v')$. Set Σ' contains this TGD up to redundancy, so by Definition 5.1 there exist a full TGD $\beta'' \rightarrow H' \in \Sigma'$ and a substitution σ such that $\sigma(\beta'') \subseteq T_k(v')$ and $\sigma(H') = G$.

- Fact G is the output of a loop at vertex v'. But then, this loop is shorter than $T_i, \ldots, T_j$ so, by property (♦), there exists a full TGD $\beta'' \rightarrow H' \in \Sigma'$ and a substitution σ such that $\sigma(\beta'') \subseteq T_k(v')$ and $\sigma(H') = G$.

Either way, the width of rule $\beta'' \rightarrow H'$ is bounded by $\mathrm{width}(\Sigma)$, and we can assume that $\beta'' \rightarrow H'$ is of the form $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H'$ where $\sigma(A'_\ell) \in T_k(v') \setminus T_{i+1}(v')$ for each $1 \leq \ell \leq n$, and $\sigma(\beta') \subseteq T_{i+1}(v')$. By property (◊), for each $1 \leq \ell \leq n$ there exist a full TGD $\beta_\ell \rightarrow H_\ell \in \Sigma'$ and a substitution σ_ℓ such that $\sigma_\ell(\beta_\ell) \subseteq T_{i+1}(v')$ and $\sigma_\ell(H_\ell) = \gamma(A'_\ell)$. Moreover, set $\mathrm{rng}(\sigma_\ell)$ clearly contains at most $\mathrm{hwidth}(\Sigma) + |\mathrm{consts}(\Sigma)|$ distinct terms. But then, there exist substitutions $\theta_1, \ldots, \theta_n$ that allow us to iteratively compose each $\beta_\ell \rightarrow H_\ell$ with $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H'$ to obtain a full TGD subsumed by some $\tau \in \Sigma'$ and substitution σ_1 such that τ and σ_1 satisfy property (◊).

To complete the proof, consider an arbitrary fact $F \in T_{j-1}(v') \setminus T_{i+1}(v')$ that is propagated from v' to v in T_j, and let $\beta'' \rightarrow H' \in \Sigma'$ and σ be the TGD and substitution whose existence is guaranteed by property (◊). Now if $\sigma(\beta'') \subseteq T_{i+1}(v') \setminus \sigma'_0(\eta_0)$, then TGD $\beta'' \rightarrow H'$ satisfies property (♦). Otherwise, we can assume that the rule is of the form $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H'$ where $\sigma(A'_i) \in \sigma'_0(\eta_0)$ for each $1 \leq i \leq n$, and $\sigma(\beta') \subseteq T_{i+1}(v') \setminus \sigma'_0(\eta_0)$. Moreover, $\mathrm{rng}(\sigma)$ clearly contains at most $\mathrm{hwidth}(\Sigma) + |\mathrm{consts}(\Sigma)|$ distinct terms. But then, there exists a substitution θ that allows us to apply the (PROPAGATE) variant of the FullDR inference rule to $\beta_0 \rightarrow \exists \vec{y}\, \eta_0$ and $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H'$ to obtain a full TGD subsumed by some TGD $\tau \in \Sigma'$ and substitution σ_1 such that τ and σ_1 satisfy property (♦). □

Proof of Complexity. The proof is analogous to the proof of complexity of Theorem 5.8. In particular, the (PROPAGATE) variant of the FullDR inference rule is analogous to the ExbDR inference rule, so we can bound in the same way the number of candidate rule pairs and possible ways to match body atoms of τ' to head atoms of τ by 32^{ℓ^2}, where $\ell = r \cdot (w + c)^a$. Once a candidate pair of τ and τ' has been selected, we need to consider all possible substitutions θ. Each such θ is defined on at most $2w$ variables $\vec{x} \cup \vec{z}$. Moreover, each variable is mapped to one of the $w + c$ variables or to one of the c constants in $consts(\Sigma)$. Hence, there are at most $(w + 2c)^{2w} \leq 4^{(w+2c)\cdot w} \leq 4^{(w+c)^2}$ different substitutions θ. Consequently, the (PROPAGATE) variant of the FullDR inference rule can be applied at most $32^{\ell^2} \cdot 4^{(w+c)^2} < 32^{n \cdot (w+c)^{2a+1}}$ times. Applications of the (COMPOSE) variant can be bounded analogously. Finally, the times needed for subsumption checking, unification, and all other steps can be bounded analogously as in the complexity proof of Theorem 5.8, with a minor difference that only body atoms need to be matched in the subsumption checks. □

The FullDR algorithm has several obvious weak points. First, it considers all possible ways to compose Datalog rules as long as this produces a rule with at most $hwidth(\Sigma) + |consts(\Sigma)|$ variables. This may seem unnecessary, but the (COMPOSE) variant of the FullDR inference rule cannot be simply dropped while retaining completeness. To understand why, consider an arbitrary loop $T_i, \ldots, T_j$ at vertex v with output fact F in a one-pass chase proof: the (PROPAGATE) variant of the FullDR inference reflects only the chase step that derives the loop's output F, so the (COMPOSE) variant is needed to reflects the chase steps that produce facts derived in the child of v that are not propagated back to v. Second, it is not clear how one efficiently obtains the atoms $A_1, \ldots, A_n$ and $A'_1, \ldots, A'_n$ participating in the (PROPAGATE) variant. Third, the number of substitutions θ in the (COMPOSE) and (PROPAGATE) variants of the FullDR inference rule can be very large. Example E.3 illustrates this problem for the (COMPOSE) variant, but one can show analogously that the (PROPAGATE) variant suffers from the same issues.

Example E.3. Consider the steps of the FullDR algorithm on GTGDs (46)–(48).

$$R(x_1, x_2) \rightarrow \exists y_1, y_2 \; S(x_1, x_2, y_1, y_2) \land T(x_1, x_2, y_2) \tag{46}$$

$$S(x_1, x_2, x_3, x_4) \rightarrow U(x_4) \tag{47}$$

$$T(z_1, z_2, z_3) \land U(z_3) \rightarrow P(z_1) \tag{48}$$

The (COMPOSE) variant of the FullDR inference rule should be applied to GTGDs (47) and (48), but it is not clear which unifier θ, identifying variables z_i in the latter with variables x_i in the former, one should use. The standard resolution inference rule from first-order theorem proving would consider only the MGU θ that maps z_3 to x_4; however, this would produce the resolvent $S(x_1, x_2, x_3, x_4) \land T(z_1, z_2, x_4) \rightarrow P(z_1)$ containing more than $hwidth(\Sigma) = 4$ variables, so this rule is not derived by the (COMPOSE) variant. Eliminating the upper bound on the number of variables is not a solution: doing so would allow the derivation of full TGDs with an unbounded number of variables, which would prevent termination. Instead, the (COMPOSE) variant requires us to consider every possible substitution θ that maps variables $x_1, \ldots, x_4, z_1, \ldots, z_3$ to at most $hwidth(\Sigma)$ variables. Consequently, $7^4 = 2401$ substitutions deriving rules such as

$$S(x_1, x_2, x_3, x_4) \land T(x_1, x_2, x_4) \rightarrow P(x_1), \tag{49}$$

$$S(x_1, x_2, x_3, x_4) \land T(x_2, x_1, x_4) \rightarrow P(x_2), \tag{50}$$

$$S(x_1, x_2, x_3, x_4) \land T(x_1, x_3, x_4) \rightarrow P(x_1), \tag{51}$$

$$S(x_1, x_2, x_3, x_4) \land T(x_3, x_1, x_4) \rightarrow P(x_3), \tag{52}$$

need to be considered, which is clearly infeasible in practice. ◄

Nevertheless, we implemented FullDR using the subsumption and indexing techniques described in Section 6. Unsurprisingly, we did not find FullDR competitive in our experiments. In fact, FullDR timed out on 173 ontologies, and there are only three ontologies where another algorithm reached the timeout but FullDR did not. For this reason, we do not discuss the results with FullDR in Section 7.