# Rewriting the Infinite Chase for Guarded TGDs

MICHAEL BENEDIKT, Oxford University, United Kingdom
MAXIME BURON, LIMOS, University Clermont Auvergne, France
STEFANO GERMANO, Oxford University, United Kingdom
KEVIN KAPPELMANN, Technical University of Munich, Germany
BORIS MOTIK, Oxford University, United Kingdom

*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 a set of GTGDs is transformed to a Datalog program that entails the same base facts on each base instance. We show that a rewriting consists of "shortcut" rules that circumvent certain chase steps, we present several algorithms that compute a rewriting by deriving such "shortcuts" efficiently, 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.

CCS Concepts: • **Information systems → Database design and models**.

Additional Key Words and Phrases: guarded database dependencies, Datalog rewriting

**ACM Reference Format:**
Michael Benedikt, Maxime Buron, Stefano Germano, Kevin Kappelmann, and Boris Motik. 2023. Rewriting the Infinite Chase for Guarded TGDs. *ACM Trans. Datab. Syst.* 1, 1, Article 1 (January 2023), 44 pages. https://doi.org/XXXXXXX.XXXXXXX

## 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 $\Sigma$, find all the answers to $Q$ that logically follow from $I$ and $\Sigma$. This problem has long been seen as a key component of a declarative data integration systems [33, 41], and it also arises in answering queries using views and accessing data sources with restrictions [27, 32, 45].

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 $\Sigma$. To answer a query, one can compute $I'$ using the chase and then evaluate the query in $I'$.

Authors' Contact Information: Michael Benedikt, Oxford University, Oxford, United Kingdom, michael.benedikt@cs.ox.ac.uk; Maxime Buron, LIMOS, University Clermont Auvergne, Aubière, France, maxime.buron@uca.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.


ACM 1557-4644/2023/1-ART1
https://doi.org/XXXXXXX.XXXXXXX

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* [28] 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)* [8] can be translated directly into GTGDs. Example 1.1 illustrates the use of GTGDs used in a 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 [29]. 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).

$$\mathsf{ACEquipment}(x) \rightarrow \exists y\, \mathsf{hasTerminal}(x, y) \wedge \mathsf{ACTerminal}(y) \tag{1}$$

$$\mathsf{ACTerminal}(x) \rightarrow \mathsf{Terminal}(x) \tag{2}$$

$$\mathsf{hasTerminal}(x, z) \wedge \mathsf{Terminal}(z) \rightarrow \mathsf{Equipment}(x) \tag{3}$$

$$\mathsf{ACTerminal}(x) \rightarrow \exists y\, \mathsf{partOf}(x, y) \wedge \mathsf{ACEquipment}(y) \tag{4}$$

Data integration is then achieved by populating the vocabulary using *mappings*, which can be as 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 $\mathsf{sw}_1$ and $\mathsf{sw}_2$, while another data source provides information about connected terminals only for switch $\mathsf{sw}_1$.

$$\mathsf{ACEquipment}(\mathsf{sw}_1) \quad \mathsf{ACEquipment}(\mathsf{sw}_2) \tag{5}$$

$$\mathsf{hasTerminal}(\mathsf{sw}_1, \mathsf{trm}_1) \quad \mathsf{ACTerminal}(\mathsf{trm}_1) \tag{6}$$

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 $\mathsf{sw}_1$ and $\mathsf{sw}_2$ 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 [42]. 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 and worst-case optimal, this method is not amenable to practical use: building the automaton and testing its emptiness are both complex and expensive, and moreover the best-case and the worst-case complexities of such algorithms typically coincide. Alternatively, one can use *blocking* to identify a tree prefix sufficient for query evaluation. Blocking is commonly used in description logic reasoning [8], and it was later lifted to *guarded logic* [34]. 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*. The idea was initially proposed by Marnette [43], and it was later extended to broader classes of TGDs [12, 31] and settings [13]. The aim is to transform a set of GTGDs $\Sigma$ into a set $\mathsf{rew}(\Sigma)$ of *Datalog*

rules such that $\Sigma$ and $\mathsf{rew}(\Sigma)$ entail the same base facts on each base instance. Thus, given a base instance $I$, instead of computing the chase of $I$ and $\Sigma$ (which may not terminate), we compute the chase $I'$ of $I$ and $\mathsf{rew}(\Sigma)$. Datalog rules essentially correspond to existential-free TGDs, so $I'$ is always finite and can be computed using optimized Datalog engines. Moreover, $\Sigma$ and $\mathsf{rew}(\Sigma)$ 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 often 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).

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

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 [35, 36], but practical algorithms for GTGDs have not yet been proposed. 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 [36] 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 very 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 $\mathsf{rew}(\Sigma)$ usually requires extending $\Sigma$ with certain logical consequences of $\Sigma$. 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 [36, 46]. *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 present an extensive account of Datalog rewriting for GTGDs. Specifically, we develop a theoretical framework that allows us to understand, motivate, and show completeness of rewriting algorithms. Moreover, we present several concrete algorithms, establish their worst-case complexity bounds, and discuss how different algorithms relate to each other. We

complement our 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 and provide insights into the performance of different algorithms. Our implementation, details about our experimental setup, and our test data are provided online [19].

*Earlier publications based on this work.* An extended abstract of this paper appeared in the VLDB proceedings [18], and a short summary of the results was included into the informal proceedings of the Datalog 2.0 workshop [17]. This paper complements these earlier publications in two main ways. First, we include the full proofs of a number of nontrivial results, such as the completeness proof of the one-pass chase and the completeness and complexity proofs of all rewriting algorithms. Second, in addition to the algorithms we presented earlier, we introduce in this paper the FullDR rewriting algorithm and we include it in our performance evaluation.

## 2 Related Work

Answering queries via rewriting has been extensively considered in description logics (DLs). For example, queries over ontologies in the DL-Lite family of languages can be rewritten into first-order queries [24]. Fact entailment can be rewritten to disjunctive Datalog for $\mathcal{SHIQ}$ ontologies [36], and to ordinary Datalog for Horn-$\mathcal{SHIQ}$ ontologies [25]. Concept subsumption algorithms for the $\mathcal{EL}$ family [7] of DLs can also be seen computing Datalog rewritings. These techniques provide the foundation for the Ontop [23], KAON2 [46], and ELK [40] systems.

In the context of TGDs, first-order rewritings were considered in data integration systems with inclusion and key dependencies [22], and frontier-guarded TGDs [15]. Datalog rewritings have been considered for GTGDs [43], frontier-guarded TGDs [13, 14], and nearly frontier-guarded and nearly guarded TGDs [31]. The focus in these studies was to identify complexity bounds and characterize expressivity of TGD classes rather than provide practical algorithms. Existing implementations of query answering for TGDs are based on first-order rewriting for linear TGDs [56], chase variants for TGDs with terminating chase [20], chase with blocking for warded TGDs [16], chase with the magic sets transformation for shy TGDs [4], and Datalog rewriting for separable and weakly separable TGDs [57]. All of these TGD classes are different from GTGDs, and we are unaware of any attempts to implement and evaluate algorithms for query answering over GTGDs.

Our algorithms are related to resolution-based decision procedures for variants of guarded logics [26, 30, 58]. Moreover, our characterization of Datalog rewritings is related to a chase variant used to answer queries over data sources with access patterns [5]. Finally, a variant of the one-pass chase from Section 4 was generalized to the broader context of disjunctive GTGDs [38].

## 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 $\alpha$ a formula or a set thereof, $\mathsf{consts}(\alpha)$, $\mathsf{vars}(\alpha)$, $\mathsf{nulls}(\alpha)$, and $\mathsf{terms}(\alpha)$ are the sets of constants, free variables, labeled nulls, and terms, respectively, in $\alpha$.

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 $\gamma$ a conjunction of facts and $I$ an instance, $\gamma \subseteq I$ means that each conjunct of $\gamma$ is contained $I$.

A *tuple generating dependency* (TGD) is a first-order formula of the form $\forall\vec{x}[\beta \rightarrow \exists\vec{y}\ \eta]$, where $\beta$ and $\eta$ are conjunctions of atoms, $\eta$ is not empty, the free variables of $\beta$ are $\vec{x}$, and the free variables of $\eta$ are contained in $\vec{x} \cup \vec{y}$. Conjunction $\beta$ 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. A TGD of the form $\forall\vec{x}[\beta \rightarrow \exists\vec{y}\ \eta \wedge A]$ where $A$ is an atom with $\mathsf{vars}(A) \cap \vec{y} = \emptyset$ is equivalent to two TGDs $\forall\vec{x}[\beta \rightarrow \exists\vec{y}\ \eta]$ and $\forall\vec{x}[\beta \rightarrow A]$; thus, each TGD can be transformed to an equivalent set of TGDs in head-normal form by repeatedly applying this transformation. 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 $\Sigma$, written $I, \Sigma \models F$, if $F \in I'$ holds for each instance $I' \supseteq I$ that satisfies $\Sigma$.

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

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

By adapting the reasoning techniques for guarded logics [6, 55] and referential database constraints [37], 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 $\tau = \forall\vec{x}[\beta \rightarrow \exists\vec{y}\ \eta]$ in head-normal form. The precondition is that there exist a vertex $v$ in $T$ and a substitution $\sigma$ with domain $\vec{x}$ such that $\sigma(\beta) \subseteq T(v)$. The result of the chase step is obtained as follows.
  - If $\tau$ is full (and thus $\eta$ 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 $\tau$ is not full, then $\sigma$ is extended to a substitution $\sigma'$ 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 set $S$ satisfying
$$\emptyset \subsetneq S \subseteq \{F \in T(v) \mid F \text{ is } \Sigma\text{-guarded by } T(v')\}.$$

A *tree-like chase sequence* for a base instance $I$ and a finite set of GTGDs $\Sigma$ 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 $\Sigma$. 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 $\Sigma$ (e.g., [42]). 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 [42].

*Rewriting.* A *Datalog rewriting* of a finite set of TGDs $\Sigma$ is a Datalog program $\text{rew}(\Sigma)$ 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 $\Sigma$ contains GTGDs only, then a Datalog rewriting $\text{rew}(\Sigma)$ 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, 46]. For example, given a base instance $I$, we can compute the *materialization* of $\text{rew}(\Sigma)$ on $I$ by applying the rules of $\text{rew}(\Sigma)$ to $I$ up to a fixpoint. This will compute precisely all base facts entailed by $\text{rew}(\Sigma)$ (and thus also by $\Sigma$) 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, as we detail next. It will be clear from the context which definitions are used 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; note that labeled nulls are not allowed to occur in terms in this case. 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 $\beta$ 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 $\sigma$ 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 $\tau$ and $y$. Then, the Skolemization of $\tau$ produces rules $\forall \vec{x}[\beta \rightarrow \sigma(H)]$ for each atom $H \in \eta$. Moreover, the Skolemization $\Sigma'$ of a finite set of TGDs $\Sigma$ 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 $\theta$ such that $\theta(A_i) = \theta(B_i)$ for $1 \leq i \leq n$. Such $\theta$ is a *most general unifier* (MGU) if, for each unifier $\sigma$ of $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$, there exists a substitution $\rho$ 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 sizes of the encoding of $A_i$ and $B_i$ [49, 50].

## 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.

Towards this goal, in Subsection 4.1 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, in Subsection 4.2 we identify the parts of such proofs that we need to be able to circumvent using "shortcuts", and we present sufficient conditions that guarantee completeness of rewriting algorithms. Finally, in Subsection 4.3 we present the proof of the main result of this section.

### 4.1 Tree-Like Chase Proofs

We start by describing formally the structure of tree-like chase proofs.

**Definition 4.1.** *A tree-like chase sequence $T_0, \dots, T_n$ for a base instance $I$ and a finite set of GTGDs $\Sigma$ 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 $T_{i-1}$:*

- *a propagation step copying exactly one fact from $v$ to its parent, or*
- *a chase step with a GTGD from $\Sigma$ provided that no propagation step from $v$ to its parent 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, a one-pass proof exists as well. The proof of Theorem 4.2 is lengthy, so we defer it to Subsection 4.3.

Theorem 4.2. *For each base instance $I$, each finite set of GTGDs $\Sigma$ 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 $\Sigma$.*

Example 4.3 illustrates important aspects of Definition 4.1 and Theorem 4.2.

Example 4.3. Let $I = \{A(a, b)\}$ and let $\Sigma$ 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 $\Sigma$ is shown in Figure 1, and it provides a proof of the base fact $H(a)$ from $I$ and $\Sigma$. 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 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 $v_1$.

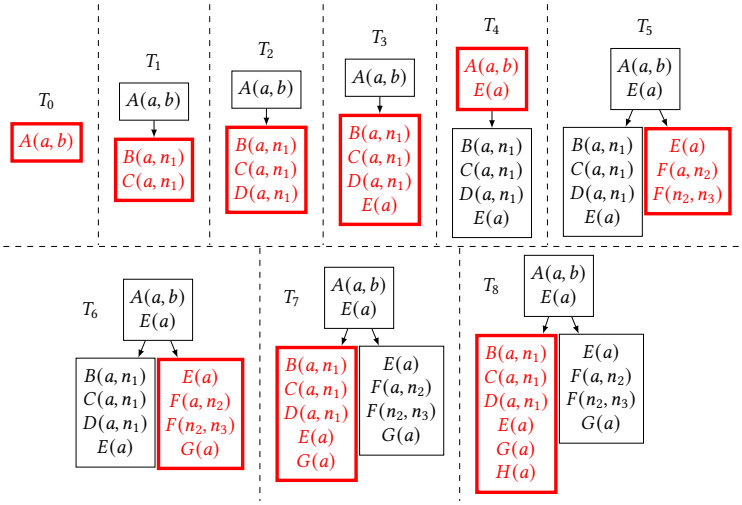



Fig. 1. Tree-Like Chase Sequence for Example 4.3

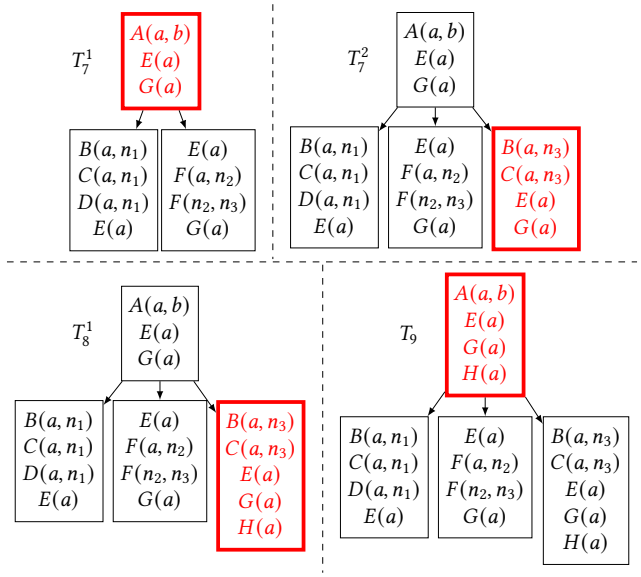



Fig. 2. One-Pass Chase Sequence Obtained from Figure 1

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* [42]: 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 $\Sigma$. Instead, we are interested only in *chase proofs*, which must only contain steps needed to demonstrate entailment of a specific fact. ◁

### 4.2 Shortcutting Loops in One-Pass Chase Proofs

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

**Definition 4.4.** *Let $T_0, \ldots, T_n$ be a one-pass tree-like chase sequence for some base instance $I$ and a finite set of GTGDs $\Sigma$ in head-normal form. A* loop at vertex $v$ with output fact $F$ *is a subsequence $T_i, \ldots, T_j$ with $0 \leq i < j \leq n$ 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 vertex $r$ with output fact $G(a)$. Finally, $T_7^1, T_7^2, T_8^1, T_9$ is a loop at vertex $r$ 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 $\Sigma$ 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 $\Sigma'$ is a rewriting of a finite set of GTGDs $\Sigma$ in head-normal form if*

- *$\Sigma'$ is a logical consequence of $\Sigma$,*
- *each Datalog rule of $\Sigma$ is a logical consequence of $\Sigma'$, and*

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

Proof. Let $\Sigma$ and $\Sigma'$ be as specified in the proposition, let $I$ be an arbitrary base instance, and let $F$ be an arbitrary base fact. Since $\Sigma'$ is a logical consequence of $\Sigma$, 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 $T_0, \dots, T_n$ of $F$ from $I$ and $\Sigma$. 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 $\overline{T}_0, \dots, \overline{T}_k$ for $I$ and $\Sigma'$ 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 $\ell_0, \dots, \ell_m$ 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 $\ell_j$ 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 $\overline{T}_{\ell_0}, \dots, \overline{T}_{\ell_{j-1}}$. 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 $\beta \to H \in \Sigma$ producing a fact $G \in T_{i_j}(r)$. The second condition of the proposition ensures that $\beta \to H$ is a logical consequence of $\Sigma'$, so $G$ can be derived from $\overline{T}_{\ell_1}(r)$ and the Datalog rules of $\Sigma'$ using $\wp$ steps. We then define $\ell_j = \ell_{j-1} + \wp$, and we append the corresponding steps to obtain the sequence $\overline{T}_0, \dots, \overline{T}_{\ell_{j-1}}, \dots, \overline{T}_{\ell_j}$.
- Otherwise, $T_{i_{j-1}}, \dots, T_{i_j}$ 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 $\beta \to H \in \Sigma'$ and a substitution $\sigma$ 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 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 $\beta \to H \in \Sigma'$ to the root vertex $r$. Moreover, $T_{i_j}(r) \subseteq \overline{T}_{\ell_j}(r)$ clearly holds, as required. □

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

We finally present a property that will be needed in the correctness proofs of the algorithms we present in Section 5. Intuitively, this property ensures that, as soon as a fact $F$ is derived in the child vertex of a loop such that $F$ does not contain any labeled nulls introduced by the child, the loop is completed and fact $F$ is propagated to the parent vertex.

Proposition 4.8. *For each loop $T_i, \dots, T_j$ at a vertex $v$ in a one-pass tree-like chase proof for some $I$ and $\Sigma$, for $i < k \leq j$, and for $v'$ the vertex introduced in $T_{i+1}$, the set $\mathsf{terms}\big(T_k(v')\big) \setminus \mathsf{nulls}\big(T_{i+1}(v')\big)$ is $\Sigma$-guarded by $T_i(v)$.*

Proof. Consider an arbitrary loop $T_i, \dots, T_j$ at a vertex $v$ in a one-pass tree-like chase proof for some base instance $I$ and a finite set of GTGDs $\Sigma$ in head-normal form, 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 $\Sigma$-guarded by $T_k(v')$, so $\mathsf{terms}(F) \subseteq \mathsf{terms}\big(T_k(v')\big) \cup \mathsf{consts}(\Sigma)$ holds.
- Assume $F$ is obtained by applying a full GTGD $\tau \in \Sigma$ to $T_k(v')$ using a substitution $\sigma$. Then, $\tau$ contains a guard atom $A$ in the body such that $\sigma(A) \subseteq T_k(v')$; moreover, the head $\tau$ contains all variables of $A$, and so we have $\mathsf{terms}(F) \subseteq \mathsf{terms}\big(T_k(v')\big) \cup \mathsf{consts}(\Sigma)$.

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

### 4.3 Proof of Theorem 4.2

We now prove Theorem 4.2. For convenience, we recapitulate the theorem's statement below.

Theorem 4.2. *For each base instance $I$, each finite set of GTGDs $\Sigma$ 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 $\Sigma$.*

Throughout this section, we fix an arbitrary base instance $I$ and a finite set of GTGDs $\Sigma$ in head-normal form. It is known that $I, \Sigma \models F$ if and only if there exists a tree-like chase proof of $F$ from $I$ and $\Sigma$. We prove Theorem 4.2 by showing that each such proof can be transformed to a one-pass chase proof of $F$ from $I$ and $\Sigma$. This argument was developed jointly with Antoine Amarilli, and it is related to proofs by Amarilli and Benedikt [5] and Kappelmann [38]; however, note that Definition 4.1 imposes slightly stronger conditions on one-pass chase sequences than the 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 [21]. The second claim captures the idea that, as the chase progresses, facts may be added within a vertex, but this will not produce new guarded sets of terms.

Lemma 4.9. *Let $T_0, \ldots, T_n$ be an arbitrary tree-like chase sequence for $I$ and $\Sigma$.*

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 $\Sigma$-guarded by both $T_i(v_1)$ and $T_i(v_2)$, and each vertex $v_3$ on the unique path in $T_i$ between $v_1$ and $v_2$, set $G$ is $\Sigma$-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 $\Sigma$-guarded by $T_i(v)$, and each $0 \leq j \leq i$ such that $T_j$ contains $v$, set $G$ is $\Sigma$-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 $\Sigma$-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, $\tau$ is guarded, so $T_i(v)$ contains a fact $S(\vec{u})$ such that $G \subseteq \vec{u}$. But then, $G$ is $\Sigma$-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 $\Sigma$-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 $\Sigma$-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 $\Sigma$-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 $\Sigma$-guarded by $F'$. Thus, each set of ground terms $G$ that is $\Sigma$-guarded by $T_{i+1}(v)$ is also $\Sigma$-guarded by $F' \in T_i(v)$, so the claim holds. □

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

**Definition 4.10.**

- *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, 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 4.11.

**Definition 4.11.** *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 4.12, 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 4.13, 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 4.14, 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 4.15, 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 4.12. *For each tree-like chase sequence $T_0, \ldots, T_n$ for $I$ and $\Sigma$, there exists a local tree-like chase sequence $\overline{T}_0, \ldots, \overline{T}_m$ for $I$ and $\Sigma$ 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 4.9, each propagation step that copies a fact $F$ between vertices $v$ and $v'$ that are further apart can be "expanded" into steps that propagate $F$ to all vertices on the unique path between $v$ and $v'$. □

Lemma 4.13. *For each local tree-like chase sequence $T_0, \ldots, T_n$ for $I$ and $\Sigma$, there exists a rootward tree-like chase sequence $\overline{T}_0, \ldots, \overline{T}_m$ for $I$ and $\Sigma$ 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 $\sigma$ and $\sigma'$, vertex $v$ is introduced into some $\overline{T}_k$ with $0 \leq k \leq m$ by a chase step with the same $\tau$, $\sigma$, and $\sigma'$.*

Proof. Let $T_0, \ldots, T_n$ be an arbitrary local tree-like chase sequence for $I$ and $\Sigma$. We prove the claim by induction on $0 \leq i \leq n$. 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 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 4.11, 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 $\Sigma$-guarded by $T_i(v')$. But then, $T_i(v') \subseteq \overline{T}_j(v')$ ensures that $F$ is also $\Sigma$-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 4.11, 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 $\tau = \beta \rightarrow \exists \vec{y}\, \eta \in \Sigma$, and let $\sigma$ and $\sigma'$ be substitutions used in the step. By the inductive property (S2), there exists $\ell_0$ with $0 < \ell_0 \leq j$ such that $v'$ is introduced in $\overline{T}_{\ell_0}$ as the result of applying a chase step with the same non-full TGD $\tau$ and substitutions $\sigma$ and $\sigma'$. 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 $\Sigma$-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 $\tau$ and substitutions $\sigma$ and $\sigma'$ 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. To this end, 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 $T_{i+i}(w) = T_i(w) \subseteq \overline{V}_m(w)$ 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 $\Sigma$-guarded by $T_i(v')$. By Claim 2 of Lemma 4.9, fact $F$ is also $\Sigma$-guarded by $T_k(v')$. Finally, by the definition of a chase step with a non-full TGD, fact $F$ is $\Sigma$-guarded by $\sigma'(\eta)$. But then, step (R3) ensures $F \in \overline{V}_0(v') \subseteq \overline{V}_m(v')$. Consequently, $T_{i+i}(v') \subseteq \overline{V}_m(v')$ 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 $\tau$ and substitutions $\sigma$ and $\sigma'$. 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 4.14. *For each rootward tree-like chase sequence $T_0, \ldots, T_n$ for $I$ and $\Sigma$, there exists an almost one-pass chase sequence $\overline{T}_0, \ldots, \overline{T}_m$ for $I$ and $\Sigma$ 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 $\Sigma$. 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 induction assumption, there exists an almost one-pass chase sequence $\overline{T}_0, \ldots, \overline{T}_j$ for $I$ and $\Sigma$ 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 $\sigma$ and $\sigma'$, 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 $\overline{T}_j$. 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 $\tau$ and substitutions $\sigma$ and $\sigma'$. 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 $F \notin \overline{T}_k(v')$ 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 $\Sigma$-guarded by $T_i(v')$. But then, $T_i(v') \subseteq \overline{T}_j(v')$ ensures that $F$ is also $\Sigma$-guarded by $\overline{T}_j(v')$, and Claim 2 of Lemma 4.9 ensures that $F$ is $\Sigma$-guarded by $\overline{T}_k(v')$. Thus, the propagation step is applicable to vertices $v$ and $v'$ in $\overline{T}_k$.

Moreover, let $\overline{U}_1, \dots, \overline{U}_s$ be the chase trees obtained by propagating $F$ starting from $\overline{U}_0$ towards the root using local steps as long as possible. Clearly, $\overline{T}_0, \dots, \overline{T}_k, \overline{U}_0, \overline{U}_1, \dots, \overline{U}_s$ 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 the sequence $\overline{T}_0, \dots, \overline{T}_j$ 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 $\overline{T}_0, \dots, \overline{T}_k, \overline{U}_0, \overline{U}_1, \dots, \overline{U}_s$.

Let $\ell$ be the smallest integer such that $\overline{T}_\ell$ contains $v''$. Clearly, $\ell \leq k$ holds. Let $N$ be the set of labeled nulls introduced by applying a chase step to $v''$ or a descendant thereof, and let $W$ the set of descendants of $v''$ in the sequence $\overline{T}_0, \dots, \overline{T}_j$. Also, let $\overline{T}'_0, \dots, \overline{T}'_k, \overline{U}'_0, \overline{U}'_1, \dots, \overline{U}'_s$ be the chase sequence obtained by uniformly replacing in $\overline{T}_0, \dots, \overline{T}_k, \overline{U}_0, \overline{U}_1, \dots, \overline{U}_s$ 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 $\overline{T}_{\ell+1}, \dots, \overline{T}_j$ into chase tress $\overline{V}_{\ell+1}, \dots, \overline{V}_j$ 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.

(T1) The chase tree $\overline{V}_p$ contains the union of the vertices of $\overline{U}'_s$ and $\overline{T}_p$.

(T2) For each vertex $w$ occurring only in $\overline{U}'_s$ (resp. $\overline{T}_p$), let $\overline{V}_p(w) = \overline{U}'_s(w)$ (resp. $\overline{V}_p(w) = \overline{T}_p(w)$).

(T3) For each vertex $w$ occurring in both $\overline{U}'_s$ and $\overline{T}_p$, let $\overline{V}_p(w) = \overline{U}'_s(w) \cup \overline{T}_p(w)$.

(T4) 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 $\tau = \forall \vec{x}[\beta \rightarrow \exists \vec{y}\, \eta]$ and substitutions $\sigma$ and $\sigma'$, 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 $\Sigma$-guarded by $\sigma'(\eta)$.

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

- Sequence $\overline{T}'_0, \dots, \overline{T}'_k, \overline{U}'_0, \overline{U}'_1, \dots, \overline{U}'_s$ 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 (T1)–(T3) clearly ensure $T_i \subseteq \overline{V}_j$. Moreover, $T_{i+1}$ differs from $T_i$ only in vertex $v'$, where $T_{i+1}(v') = T(v') \cup \{F\}$. Our construction

clearly ensures $F \in \overline{U}'_s(v'')$, and step (T4) 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 4.15. *For each base fact $F$ and each almost one-pass tree-like chase proof of $F$ from $I$ and $\Sigma$, there exists a one-pass tree-like chase proof of $F$ from $I$ and $\Sigma$.*

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 $\Sigma$. 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, in each remaining $T_j$, we delete 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 of the sequence, the sequence is a tree-like chase proof of $F$ from $I$ and $\Sigma$. □

## 5 Rewriting Algorithms

In this section we consider several possibilities for producing "shortcut" Datalog rules that satisfy Proposition 4.7. In particular, in Subsection 5.1 we present the FullDR algorithm that manipulates GTGDs, but derives only Datalog rules. We also present three algorithms that can produce intermediate GTGDs/rules outside Datalog: in Subsection 5.2 we present the ExbDR algorithm that manipulates GTGDs directly, and in Subsections 5.3 and 5.4 we present the SkDR and HypDR algorithms, respectively, that manipulate rules obtained by Skolemizing the input GTGDs.

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 $\Sigma$, the algorithm applies Inf to (the Skolemization of) $\Sigma$ 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 $\tau_1$* subsumes *a TGD $\tau_2 = \forall \vec{x}_2[\beta_2 \rightarrow \exists \vec{y}_2\ \eta_2]$ if there exists a substitution $\mu$ such that $\mathsf{dom}(\mu) = \vec{x}_1 \cup \vec{y}_1$, $\mu(\vec{x}_1) \subseteq \vec{x}_2$, $\mu(\vec{y}_1) \subseteq \vec{y}_1 \cup \vec{y}_2$, $\mu(y) \neq \mu(y')$ 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 $H_1 \in \beta_1$. Moreover, rule $\tau_1$* subsumes *a rule $\tau_2 = \forall \vec{x}_2[\beta_2 \rightarrow H_2]$ if there exists a substitution $\mu$ such that $\mu(\beta_1) \subseteq \beta_2$ and $\mu(H_1) = H_2$.*

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

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.

Our definitions of syntactic tautologies and rule subsumption are the same as in first-order theorem proving [10]. For example, consider the following rules.

$$A(f(x_1), f(x_1)) \wedge B(x_1) \rightarrow B(f(x_1)) \tag{17}$$

$$A(x_2, x_3) \rightarrow B(x_2) \tag{18}$$

Rule (17) is subsumed by rule (18) using substitution $\mu_1$ that maps both $x_2$ and $x_3$ to $f(x_1)$. This ensures that, if rule (17) derives $B(f(t))$ in one step from a set of facts $I$ by a substitution $\sigma$ where $\sigma(x_1) = t$, then rule (18) also derives $B(f(t))$ from $I$ in one step by substitution $\sigma \circ \mu_1$. Thus, rule (17) is not needed when rule (18) is present, so rule (17) can be discarded.

In contrast, our notion of TGD subsumption is not standard and is slightly more involved. To illustrate the details, consider the following TGDs.

$$A(x_1, x_1) \wedge B(x_1) \rightarrow \exists y_1\, C(x_1, y_1) \tag{19}$$

$$A(x_2, x_3) \rightarrow \exists y_2, y_3\, C(x_2, y_2) \wedge D(x_3, y_3) \tag{20}$$

TGD (19) is subsumed by TGD (20) by substitution $\mu_2$ 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 $\mu_2$ 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 TGD (19) and substitutions $\sigma$ and $\sigma'$ can always be obtained (up to isomorphism) by a chase step with TGD (20) 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.** *Let* Inf *be an inference rule and let* $\Sigma$ *be a finite set of GTGDs. Then,* $\mathsf{Inf}(\Sigma)$ *is the smallest set that contains up to variable renaming each TGD/rule obtained by*

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

*Finally, for each* $\Sigma' \subseteq \mathsf{Inf}(\Sigma)$ *such that* $\Sigma'$ *contains* $\mathsf{Inf}(\Sigma)$ *up to redundancy, the subset of the Skolem-free Datalog rules of* $\Sigma'$ *is an* Inf*-rewriting of* $\Sigma$*.*

Since inference rule application is monotonic, set $\mathsf{Inf}(\Sigma)$ is unique up to variable renaming for each Inf and $\Sigma$. In contrast, redundancy allows dropping conclusions of certain inferences and is thus not monotonic; consequently, an Inf-rewriting is not necessarily unique. Note, however, that each set contains itself up to redundancy, so the subset of the Skolem-free Datalog rules of $\mathsf{Inf}(\Sigma)$ is an Inf-rewriting of $\Sigma$. Smaller Inf-rewritings can be obtained by interleaving inference rule application with deletion of redundant TGDs/rules. In Section 6, we present a systematic way to do so and produce a subset-minimal subset $\Sigma'$ of $\mathsf{Inf}(\Sigma)$ that contains $\mathsf{Inf}(\Sigma)$ up to redundancy, and consequently obtain a smaller Inf-rewriting of $\Sigma$.

### 5.1 Creating Datalog Rules Directly

An intuitive design objective for a rewriting algorithm is to ensure that the algorithm produces only Datalog rules, without deriving any intermediate non-full TGDs in the process. In this subsection, we present one such algorithm. Similar algorithms have appeared in the literature [5]. After presenting our algorithm, we discuss the shortcomings of such an approach.

**Definition 5.4.** *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*

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

*and a substitution* $\theta$ *such that*

- $\theta(A) = \theta(A')$,
- $\mathsf{dom}(\theta) = \vec{x} \cup \vec{z}$, *and*
- $\mathsf{rng}(\theta) \subseteq \vec{w} \cup \mathsf{consts}(\tau) \cup \mathsf{consts}(\tau')$ *where* $\vec{w}$ *is a vector of* $\mathsf{hwidth}(\Sigma) + |\mathsf{consts}(\Sigma)|$ *variables different from* $\vec{x} \cup \vec{z}$,

*and it derives*

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

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

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

*and a substitution* $\theta$ *such that*

- $\theta(A_i) = \theta(A'_i)$ *for each* $i$ *with* $1 \leq i \leq n$,
- $\mathsf{dom}(\theta) = \vec{x} \cup \vec{z}$,
- $\mathsf{rng}(\theta) \subseteq \vec{w} \cup \vec{y} \cup \mathsf{consts}(\tau) \cup \mathsf{consts}(\tau')$ *where* $\vec{w}$ *is a vector of* $\mathsf{hwidth}(\Sigma) + |\mathsf{consts}(\Sigma)|$ *variables different from* $\vec{x} \cup \vec{y} \cup \vec{z}$,
- $\theta(\vec{x}) \cap \vec{y} = \emptyset$, *and*
- $\mathsf{vars}(\theta(\beta')) \cap \vec{y} = \emptyset$ *and* $\mathsf{vars}(\theta(H')) \cap \vec{y} = \emptyset$,

*and it derives*

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

Theorem 5.5. *Each* FullDR*-rewriting of each finite set of GTGDs* $\Sigma$ *is a Datalog rewriting of* $\Sigma$. *Moreover,* $\mathsf{FullDR}(\Sigma)$ *can be computed in time* $O(b^{r^d \cdot (w+c)^{da}})$ *for* $r$ *the number of relations in* $\Sigma$, $a$ *the maximum relation arity in* $\Sigma$, $w = \mathsf{width}(\Sigma)$, $c = |\mathsf{consts}(\Sigma)|$, *and some constants* $b$ *and* $d$.

Proof of Correctness. Fix an arbitrary finite set of GTGDs $\Sigma$, fix an arbitrary $\Sigma' \subseteq \mathsf{FullDR}(\Sigma)$ that contains $\mathsf{FullDR}(\Sigma)$ up to redundancy, and let $\Sigma''$ be the set of Datalog TGDs of $\Sigma'$. It is easy to see that $\mathsf{FullDR}(\Sigma)$ is a logical consequence of $\Sigma$, so $\Sigma'$ and $\Sigma''$ are logical consequences of $\Sigma$ as well. Moreover, $\Sigma''$ contains each full GTGD of $\Sigma$ up to redundancy, so each full GTGD of $\Sigma$ is logically entailed by $\Sigma''$. We next consider an arbitrary base instance $I$ and a one-pass tree-like chase sequence for $I$ and $\Sigma$, and we show the following property:

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

Since $\Sigma''$ contains all Datalog rules of $\Sigma'$ and this property holds for the root vertex $r$, Proposition 4.7 ensures that $\Sigma''$ is a Datalog rewriting of $\Sigma$.

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 $v$, and assume that the claim holds for all shorter loops. 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 TGD $\forall \vec{x}[\beta_0 \rightarrow \exists \vec{y}\, \eta_0] \in \Sigma$. Let $\sigma_0$ and $\sigma'_0$ 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 $\mathsf{hwidth}(\Sigma) + |\mathsf{consts}(\Sigma)|$ 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 $\mathsf{width}(\Sigma)$ and a substitution $\sigma$ 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 $\Sigma$ derives $G$ from $T_k(v')$. Set $\Sigma'$ contains this TGD up to redundancy, so by Definition 5.1 there exist a full TGD $\beta'' \to H' \in \Sigma'$ and a substitution $\sigma$ 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 $\beta'' \to H' \in \Sigma'$ and a substitution $\sigma$ such that $\sigma(\beta'') \subseteq T_k(v')$ and $\sigma(H') = G$.

Either way, the width of rule $\beta'' \to H'$ is bounded by $\mathsf{width}(\Sigma)$, and we can assume that $\beta'' \to H'$ is of the form $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$ 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 $\beta_\ell \to H_\ell \in \Sigma'$ and a substitution $\sigma_\ell$ such that $\sigma_\ell(\beta_\ell) \subseteq T_{i+1}(v')$ and $\sigma_\ell(H_\ell) = \gamma(A'_\ell)$. Moreover, set $\mathsf{rng}(\sigma_\ell)$ contains at most $\mathsf{hwidth}(\Sigma) + |\mathsf{consts}(\Sigma)|$ distinct terms. But then, there exist substitutions $\theta_1, \ldots, \theta_n$ that allow us to iteratively apply the (COMPOSE) variant of the FullDR inference rule to $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$ and all $\beta_\ell \to H_\ell$ to obtain a substitution $\sigma_1$ and a full TGD that is subsumed by some $\tau \in \Sigma'$ such that $\tau$ and $\sigma_1$ 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 $\beta'' \to H' \in \Sigma'$ and $\sigma$ 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 $\beta'' \to H'$ satisfies property (♦). Otherwise, we can assume that the rule is of the form $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$ 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, $\mathsf{rng}(\sigma)$ clearly contains at most $\mathsf{hwidth}(\Sigma) + |\mathsf{consts}(\Sigma)|$ distinct terms. But then, there exists a substitution $\theta$ that allows us to apply the (PROPAGATE) variant of the FullDR inference rule to $\beta_0 \to \exists \vec{y}\, \eta_0$ and $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$ to obtain a full TGD subsumed by some TGD $\tau \in \Sigma'$ and substitution $\sigma_1$ such that $\tau$ and $\sigma_1$ satisfy property (♦). □

Proof of Complexity. Fix $\Sigma$, $r$, $w$, $c$, and $a$ as stated in the theorem. The number of different body atoms of arity $a$ constructed using $r$ relations, $w$ variables, and $c$ constants is clearly bounded by $\ell = r \cdot (w + c)^a$. The number of variables in each GTGD in the input, as well as in each TGD produced by the FullDR inference rule is bounded by $w$. Although the latter TGDs have just one head atom, the GTGDs in the input can contain several head atoms so, to simplify our calculation, we consider all TGDs as possibly having multiple head atoms. Then, the head and the body of each TGD corresponds to a subset of these $\ell$ atoms, so the number of different TGDs up to variable renaming is bounded by $\wp = 2^\ell \cdot 2^\ell$. Thus, the FullDR inference rule needs to be applied to at most $\wp^2 = 2^{4\ell}$ pairs of TGDs. To apply the (PROPAGATE) variant, for each such pair we might need to consider each possible way to match at most $\ell$ body atoms of $\tau'$ to $\ell$ head atoms of $\tau$, and there are at most $\ell^\ell \leq 2^{\ell^2}$ of these. Consequently, substitution $\theta$ may need to be computed at most $2^{4\ell} \cdot 2^{\ell^2} \leq 2^{5\ell^2} = 32^{\ell^2}$ times. Each such $\theta$ 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 $\mathsf{consts}(\Sigma)$. Hence, there are at most $(w + 2c)^{2w} \leq 4^{(w+2c)\cdot w} \leq 4^{(w+c)^2}$ different substitutions $\theta$. 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. The number of applications of the (COMPOSE) variant can be bounded analogously. Finally, all other steps require linear time. □

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 $\mathsf{hwidth}(\Sigma) + |\mathsf{consts}(\Sigma)|$ 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 child $v'$ and 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 $F$, but the derivation of $F$ in $v'$ may depend on the prior derivation of another fact $F'$ in $v'$. The (COMPOSE) variant allows us to produce $F$ in $v'$ without $F'$, rendering it eligible for (PROPAGATE) again. Second, it is not clear how to efficiently select the atoms $A_1, \ldots, A_n$ and $A'_1, \ldots, A'_n$ participating in the (PROPAGATE) variant. Third, the number of substitutions $\theta$ in the (COMPOSE) and (PROPAGATE) variants of the FullDR inference rule can be very large. Example 5.6 illustrates this for the (COMPOSE) variant, but the (PROPAGATE) variant suffers from analogous issues.

Example 5.6. Consider the derivations of the FullDR algorithm on the set of GTGDs (21)–(23).

$$R(x_1, x_2) \rightarrow \exists y_1, y_2\, S(x_1, x_2, y_1, y_2) \wedge T(x_1, x_2, y_2) \tag{21}$$

$$S(x_1, x_2, x_3, x_4) \rightarrow U(x_4) \tag{22}$$

$$T(z_1, z_2, z_3) \wedge U(z_3) \rightarrow P(z_1) \tag{23}$$

The (COMPOSE) variant of the FullDR inference rule should be applied to GTGDs (22) and (23), but it is not clear which substitution $\theta$, 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 $\theta$ that maps $z_3$ to $x_4$; however, this would produce the resolvent $S(x_1, x_2, x_3, x_4) \wedge T(z_1, z_2, x_4) \rightarrow P(z_1)$ containing more than $\mathsf{hwidth}(\Sigma) = 4$ variables, so this rule is not derived by the (COMPOSE) variant. Eliminating the upper bound on the number of variables 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 $\theta$ that maps variables $x_1, \ldots, x_4, z_1, \ldots, z_3$ to at most $\mathsf{hwidth}(\Sigma)$ variables. Consequently, $7^4 = 2401$ substitutions are considered, deriving rules

$$S(x_1, x_2, x_3, x_4) \wedge T(x_1, x_2, x_4) \rightarrow P(x_1), \tag{24}$$

$$S(x_1, x_2, x_3, x_4) \wedge T(x_2, x_1, x_4) \rightarrow P(x_2), \tag{25}$$

$$S(x_1, x_2, x_3, x_4) \wedge T(x_1, x_3, x_4) \rightarrow P(x_1), \tag{26}$$

and so on. This is the main obstacle to using the FullDR algorithm in practice. In Sections 5.2–5.4 we present three additional algorithms, neither of which exhibits such behavior on GTGDs (21)–(23). ◁

Despite these drawbacks, the FullDR algorithm is useful because it illustrates the complexities introduced by the desire to produce only Datalog rules. Moreover, as we show in Section 6, the algorithm is still able to compute a rewriting of many nontrivial inputs.

### 5.2 The Existential-Based Rewriting

We next present the ExbDR rewriting algorithm that promises to be more practical than FullDR. Unlike FullDR, ExbDR generates intermediate non-full GTGDs that are eventually discarded. We first discuss the intuition by means of the properties of the one-pass chase. In particular, in Section 4 we argued that 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 $T_k(v')$ 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 a full GTGD, so this GTGD provides us with the desired loop "shortcut". Before formalizing this idea, we slightly adapt the notion of unification.

**Definition 5.7.** *For $X$ a set of variables, an $X$-*unifier *and an $X$-*MGU *$\theta$ of atoms $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ are defined as in Section 3, but with additionally requiring that $\theta(x) = x$ for each $x \in X$.*

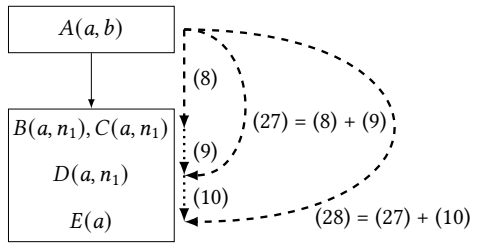



Fig. 3. Deriving "shortcuts" for the loop $T_0$–$T_4$ in ExbDR

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 5.8.** *The* Existential-Based Datalog Rewriting *inference rule* ExbDR *takes two GTGDs*

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

*with $n \geq 1$ and, for $\theta$ 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 $\mathsf{vars}(\theta(\beta')) \cap \vec{y} = \emptyset$, 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 5.9. Consider the set $\Sigma$ of GTGDs from Example 4.3; for convenience, we repeat $\Sigma$ below using the same equation numbers as before.

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

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 (27), 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 (27) and (10) to produce (28), which again derives all facts of $v_1$ from the loop's input in one step.

$$A(x_1, x_2) \rightarrow \exists y\, B(x_1, y) \wedge C(x_1, y) \wedge D(x_1, y) \tag{27}$$
$$A(x_1, x_2) \rightarrow \exists y\, B(x_1, y) \wedge C(x_1, y) \wedge D(x_1, y) \wedge E(x_1) \tag{28}$$

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 (28): atom $E(x_1)$ does not contain any existential variables. Definition 5.3 requires each derived GTGD to be brought into head-normal, so (28) is broken up into (27) 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 (29), and transformation into head-normal form produces (11) and (15). Moreover, (8) and (13) produce (30), and transformation (30) into head-normal form produces (16) and (31).

$$A(x_1, x_2) \wedge E(x_1) \rightarrow \exists y_1, y_2\ F(x_1, y_1) \wedge F(y_1, y_2) \wedge G(x_1) \tag{29}$$

$$A(x_1, x_2) \wedge G(x_1) \rightarrow \exists y\ B(x_1, y) \wedge C(x_1, y) \wedge H(x_1) \tag{30}$$

$$A(x_1, x_2) \wedge G(x_1) \rightarrow \exists y\ B(x_1, y) \wedge C(x_1, y) \tag{31}$$

GTGD (31) 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 $\Sigma$. ◁

Example 5.10. Unlike FullDR, the ExbDR algorithm not does need to explore a potentially very large number of substitutions $\theta$. Consider applying the ExbDR algorithm to GTGDs (21)–(23) from Example 5.6. The algorithm uses (21) and (22) to produce (32), and it then uses (23) and (32) to produce (33). Finally, transforming (33) into head-normal form produces (32) and (34).

$$R(x_1, x_2) \rightarrow \exists y_1, y_2\ S(x_1, x_2, y_1, y_2) \wedge T(x_1, x_2, y_2) \wedge U(y_2) \tag{32}$$

$$R(x_1, x_2) \rightarrow \exists y_1, y_2\ S(x_1, x_2, y_1, y_2) \wedge T(x_1, x_2, y_2) \wedge U(y_2) \wedge P(x_1) \tag{33}$$

$$R(x_1, x_2) \rightarrow P(x_1) \tag{34}$$

No further inferences are possible after this, so (22), (23), and (34) are returned as the rewriting. ◁

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

Proposition 5.11. *Each application of the* ExbDR *inference rule to $\tau$, $\tau'$, and $\theta$ as in Definition 5.8 satisfies the following properties.*

1. *Some atom $A'_i$ with $1 \leq i \leq n$ is a guard in $\tau'$.*
2. *For each $1 \leq i \leq n$ such that $A'_i$ is a guard of $\tau'$, and for $\sigma$ 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 $\mathsf{vars}(\sigma(A'_j)) \cap \vec{y} \neq \emptyset$ for each $1 \leq j \leq n$.*
3. *The rule produces a GTGD with body and head at most $\mathsf{bwidth}(\Sigma)$ and $\mathsf{hwidth}(\Sigma)$, respectively.*

Note that, in the second claim, $\sigma$ unifies only $A_i$ and $A'_i$, whereas $\theta$ unifies all $A_1, \ldots, A_n$ and $A'_1, \ldots, A'_n$; thus, $\sigma$ and $\theta$ are not necessarily the same.

Proof of Claim 1. Let $G$ be a guard for $\tau'$. 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 $\tau'$ is matched to $A_1$ in the head of $\tau$. Since $\tau$ 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 $\theta$ 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 $\tau'$, variable $z$ occurs in $G$. Therefore, we have $\mathsf{vars}(\theta(G)) \cap \vec{y} \neq \emptyset$, which contradicts the requirement $\mathsf{vars}(\theta(\beta')) \cap \vec{y} = \emptyset$ of the ExbDR inference rule. □

Proof of Claim 2. Consider arbitrary $i$ such that $1 \leq i \leq n$ and $A'_i$ is a guard of $\tau'$, and let $\sigma$ be an MGU of $A'_i$ and the corresponding atom $A_i$ of $\tau$. Since $\theta$ is a unifier of $A'_i$ and $A_i$ as well as of other pairs of atoms, there clearly exists a substitution $\rho$ such that $\theta = \rho \circ \sigma$. Now consider an arbitrary $A'_j$ with $1 \leq j \leq n$ in $\tau'$. Substitution $\theta$ matches $A'_j$ to the corresponding atom $A_j$ in the head of $\tau$. Since TGD $\tau$ is in head-normal form, atom $A_j$ contains at least one variable $y \in \vec{y}$. Since $\theta(y) = y$, we necessarily have $y \in \mathsf{vars}(\theta(A'_j))$. Consequently, atom $A'_j$ contains some variable $z$ such that $\theta(z) = y$. Since $A'_i$ is a guard for $\tau'$, 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 $x \in \mathsf{vars}(A_i)$ 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. □

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 $\tau'$. Thus, $\mathsf{vars}(\beta') \cup \mathsf{vars}(H') \subseteq \mathsf{vars}(A'_i)$. The ExbDR inference rule ensures $\mathsf{vars}(\theta(\beta')) \cap \vec{y} = \emptyset$, which in turn ensures $\mathsf{vars}(\theta(\beta')) \subseteq \mathsf{vars}(\theta(A'_i)) \setminus \vec{y}$. Now let $G$ be a guard for $\tau$. We clearly have $\mathsf{vars}(\theta(\beta)) \subseteq \mathsf{vars}(\theta(G))$. Moreover, $\theta(y_i) = y_i$ and $\theta(x) \cap \vec{y} = \emptyset$ ensure

$$\mathsf{vars}(\theta(\eta)) \cup \mathsf{vars}(\theta(A_i)) \subseteq \mathsf{vars}(\theta(G)) \cup \vec{y}.$$

Thus, $\theta(G)$ is a guard for the TGD produced by the ExbDR inference rule. Finally, since $G$ contains all variables of $\tau$, the widths of the resulting TGD and $\tau$ are equal. □

The third claim of Proposition 5.11 is needed to prove termination of ExbDR, and the first two claims 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 5.11 tells us that a guard of $\tau'$ will definitely participate in the inference. Thus, we can choose one such guard $G' \in \beta'$ of $\tau'$ and try to find a $\vec{y}$-MGU $\sigma$ of $G'$ and a counterpart atom $G \in \eta$ from the head of $\tau$. 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 5.11, all candidates for the atoms participating in the inference will contain a variable that is mapped by $\sigma$ to a member of $\vec{y}$; thus, $S' = \{\sigma(A') \mid A' \in \beta' \wedge \mathsf{vars}(\sigma(A')) \cap \vec{y} \neq \emptyset\}$ is the set of all relevant side atoms. Note that we apply $\sigma$ to the atoms in $S'$ to simplify further matching. The next step is to identify the corresponding head atoms of $\tau$. 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 $1 \leq i \leq n$, 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 $\theta$ of sets $S$ and $S'$. If unification succeeds, we derive the corresponding GTGD.

Theorem 5.12. *Each* ExbDR*-rewriting of each finite set of GTGDs $\Sigma$ is a Datalog rewriting of $\Sigma$. Moreover,* $\mathsf{ExbDR}(\Sigma)$ *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 $\Sigma$, $a$ the maximum relation arity in $\Sigma$, $w_b = \mathsf{bwidth}(\Sigma)$, $w_h = \mathsf{hwidth}(\Sigma)$, $c = |\mathsf{consts}(\Sigma)|$, and some constants $b$ and $d$.*

Proof of Correctness. Fix an arbitrary finite set of GTGDs $\Sigma$, fix an arbitrary $\Sigma' \subseteq \mathsf{ExbDR}(\Sigma)$ that contains $\mathsf{ExbDR}(\Sigma)$ up to redundancy, and let $\Sigma''$ be the set of Datalog TGDs of $\Sigma'$. It is easy to see that $\mathsf{ExbDR}(\Sigma)$ is a logical consequence of $\Sigma$, so $\Sigma'$ and $\Sigma''$ are logical consequences of $\Sigma$ as well. Moreover, $\Sigma''$ contains each full GTGD of $\Sigma$ up to redundancy, so each full GTGD of $\Sigma$ is logically entailed by $\Sigma''$. We next consider an arbitrary base instance $I$ and a one-pass tree-like chase sequence for $I$ and $\Sigma$, and we show the following property:

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

Since $\Sigma''$ contains all Datalog rules of $\Sigma'$ and this property holds for the root vertex $r$, Proposition 4.7 ensures that $\Sigma''$ is a Datalog rewriting of $\Sigma$.

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 $v$, and assume that the claim holds for all shorter loops. 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 TGD $\forall \vec{x}[\beta_0 \rightarrow \exists \vec{y}\, \eta_0] \in \Sigma$. Let $\sigma_0$ and $\sigma'_0$ be the substitutions used in this chase step, let $N = \mathsf{nulls}(\mathsf{rng}(\sigma'_0)) \setminus \mathsf{nulls}(\mathsf{rng}(\sigma_0))$, 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 $\Sigma$-guarded by $\sigma'_0(\eta_0)$. Thus, we have $T_{i+1}(v') = S \cup \sigma'_0(\eta_0)$. By Proposition 4.8 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 $\Sigma$-guarded by $T_i(v)$ and thus propagated back to $v$. 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 $i < k < j-1$:

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

We prove ($\Diamond$) by induction on $k$. We have already proved the base case $k = i+1$ above. For the inductive step, assume that ($\Diamond$) holds for some $k$, so there exists a GTGD $\forall\vec{x}[\beta \to \exists\vec{y}\,\eta] \in \Sigma'$ and substitutions $\sigma$ and $\sigma'$ satisfying ($\Diamond$) for $k$. Now consider $T_{k+1}$. Property ($\Diamond$) 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 $\Sigma$ derives $G$ from $T_k(v')$. Set $\Sigma'$ contains this GTGD up to redundancy, so by Definition 5.1 there exist a full GTGD $\beta'' \to H' \in \Sigma'$ and a substitution $\rho$ 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 ($\blacklozenge$), there exists a full GTGD $\beta'' \to H' \in \Sigma'$ and a substitution $\rho$ such that $\rho(\beta'') \subseteq T_k(v')$ and $\rho(H') = G$.

Since $\eta$ 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 $\beta''$ 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 $\eta$ such that $\sigma'(A_i) = \rho(A'_i)$ for $1 \leq i \leq n$. Since $\sigma'$ 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 $\theta$ 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 $\mathrm{vars}(\theta(\beta')) \cap \vec{y} = \emptyset$ 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 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 $\sigma$ and $\sigma'$ to substitutions $\zeta$ and $\zeta'$, 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 $\Sigma'$ contains $\tau$ up to redundancy. Since $G \notin T_k(v')$, GTGD $\tau$ 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 $\mu$ 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 $\mu(\beta_1) \subseteq \theta(\beta) \wedge \theta(\beta')$ and $\mu(\eta_1) \supseteq \theta(\eta) \wedge \theta(H')$. Now let $\sigma_1$ be the substitution defined on $\vec{x}$ such that $\sigma_1(x) = \zeta(\mu(x))$ on each $x \in \vec{x}$, and let $\sigma'_1$ be the extension of $\sigma_1$ 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 exist a GTGD $\forall\vec{x}[\beta \to \exists\vec{y}\,\eta] \in \Sigma'$ and substitutions $\sigma$ and $\sigma'$ such that $\sigma(\beta) \subseteq T_i(v)$ and $T_{j-2}(v') = S \cup \sigma'(\eta)$. Then, as above, $\Sigma'$ contains a full TGD of the form $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$ such that, for some substitution $\rho$, we have $\rho(A'_1) \cup \cdots \cup \rho(A'_n) \subseteq \sigma'(\eta)$ and $\rho(\beta') \subseteq S$. 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 $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H'$ to produce

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

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

Proof of Complexity. Fix $\Sigma$, $r$, $w_b$, $w_h$, $c$, and $a$ as in the theorem. The number of body atoms of arity $a$ constructed using $r$ relations, $w_b$ variables, and $c$ constants is bounded by $\ell_b = r \cdot (w_b + c)^a$. By the third claim of Proposition 5.11, 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 $\ell_b$ body atoms of $\tau'$ to $\ell_h$ head atoms of $\tau$, and there are at most $(\ell_h)^{\ell_b} \leq 2^{\ell_b \cdot \ell_h}$ of these. Consequently, unifier $\theta$ may need to be computed at most $2^{2(\ell_b + \ell_h)} \cdot 2^{\ell_b \cdot \ell_h} \leq 2^{5\ell_b \cdot \ell_h} = 32^{\ell_b \cdot \ell_h}$ times. Finally, unification of atoms requires time linear in $a$, and all other steps require linear time too. □

Program $\mathsf{ExbDR}(\Sigma)$ 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 $\mathsf{ExbDR}(\Sigma)$ is worst-case optimal. To see why, let $r$, $a$, and $c$ be as in Theorem 5.12, and consider a base instance $I$ with $c'$ constants. The fixpoint of $\mathsf{ExbDR}(\Sigma)$ on $I$ contains at most $r(c + c')^a$ facts, and it can be computed in time $O(r(c + c')^a \cdot |\mathsf{ExbDR}(\Sigma)|)$: each rule $\tau \in \mathsf{ExbDR}(\Sigma)$ is guarded so we can apply a chase step with $\tau$ by matching a guard and then checking the remaining body atoms. Hence, we can compute $\mathsf{ExbDR}(\Sigma)$ and find its fixpoint in 2ExpTime, in ExpTime if the relation arity is fixed, and in PTime if $\Sigma$ is fixed (i.e., if we consider *data complexity*). These results match the lower bounds for checking fact entailment for GTGDs [42].

### 5.3 Using Skolemization

The ExbDR algorithm exhibits two drawbacks. First, applications of the ExbDR inference rule can introduce head atoms, so the rule heads can get very long. Second, each inference requires matching a subset of body atoms of $\tau'$ to a subset of the head atoms of $\tau$; despite the optimizations outlined after Proposition 5.11, 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 $\tau'$ with the head atom of $\tau$, but this does not seem possible if we stick to manipulating GTGDs. For example, atoms $C(y)$ and $D(y)$ of GTGD (27) refer to the same labeled null (represented by variable $y$), and this information would be lost if we split (27) 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 5.14. Before presenting the algorithm, in Definition 5.13 we generalize the notion of guardedness to rules.

**Definition 5.13.** *Rule* $\forall \vec{x}[\beta \rightarrow H]$ *is* guarded *if each function symbol in the rule is a Skolem symbol, the body* $\beta$ *contains a Skolem-free atom* $A \in \beta$ *such that* $\mathsf{vars}(A) = \vec{x}$*, and each Skolem term in the rule is of the form* $f(\vec{t})$ *where* $\mathsf{vars}(f(\vec{t})) = \vec{x}$ *and* $\vec{t}$ *is function-free.*

**Definition 5.14.** *The* Skolem Datalog Rewriting *inference rule* SkDR *takes two guarded rules*

$$\tau = \beta \rightarrow H \qquad \textit{and} \qquad \tau' = A' \wedge \beta' \rightarrow H'$$

*such that*

- *$\beta$ is Skolem-free and $H$ contains a Skolem symbol, and*
- *$A'$ contains a Skolem symbol, or $\tau'$ is Skolem-free and $A'$ contains all variables of $\tau'$,*

*and, for $\theta$ an MGU of $H$ and $A'$, it derives*

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

Example 5.15. We illustrate how the SkDR inference rule is applied to the set $\Sigma$ of GTGDs from Example 4.3; again, for the sake of readability, we repeat $\Sigma$ using the same equation numbers.

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

Skolemizing GTGDs (8) and (11) produces rules (35)–(36), and (37)–(38), respectively.

$$A(x_1, x_2) \rightarrow B(x_1, f(x_1, x_2)) \tag{35}$$
$$A(x_1, x_2) \rightarrow C(x_1, f(x_1, x_2)) \tag{36}$$
$$A(x_1, x_2) \wedge E(x_1) \rightarrow F(x_1, g(x_1, x_2)) \tag{37}$$
$$A(x_1, x_2) \wedge E(x_1) \rightarrow F(g(x_1, x_2), h(x_1, x_2)) \tag{38}$$

Intuitively, rules (35) and (36) 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 (35) and (10) to obtain (39); it combines (36) and (9) to obtain (40); and it combines (39) and (40) to obtain the "shortcut" rule (14).

$$A(x_1, x_2) \wedge D(x_1, f(x_1, x_2)) \rightarrow E(x_1) \tag{39}$$
$$A(x_1, x_2) \rightarrow D(x_1, f(x_1, x_2)) \tag{40}$$

The rules with Skolem-free bodies derived in this way allow us to reconstruct derivations in one step analogously to Example 5.9, and the rules with Skolem symbols in body atoms capture the intermediate derivation steps. For example, rule (39) captures the result of matching the first body atom of rule (10) to the fact produced by rule (35). To complete the rewriting, SkDR combines (37) with (12) to obtain (15), and it combines (35) with (13) to derive (16).

However, SkDR also combines (10) and (40) into (41), which with (35) derives (14) the second time. Analogously to (39), rule (41) captures the result of matching the second body atom of rule (10) to the fact produced by rule (40). However, all of these inferences are superfluous: they just process the two body atoms of (10) in a different order. Also, SkDR combines (12) and (38) into rule (42), which is a "dead-end" in that it does not further contribute to a Datalog rule.

$$A(x_1, x_2) \wedge B(x_1, f(x_1, x_2)) \rightarrow E(x_1) \tag{41}$$
$$A(x_1, x_2) \wedge E(x_1) \wedge E(g(x_1, x_2)) \rightarrow G(g(x_1, x_2)) \tag{42}$$

Our HypDR algorithm in Subsection 5.4 can avoid these overheads, but at the expense of using more than two rules at a time. ◁

Proposition 5.16 and Theorem 5.17 capture the relevant properties of the SkDR algorithm.

Proposition 5.16. *Each application of the* SkDR *inference rule to rules $\tau$ and $\tau'$ as in Definition 5.14 produces a guarded rule.*

Proof. In our proof, we reuse the results by de Nivelle [26] about unification of atoms in guarded rules. The *variable depth* [26, 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* [26, Definition 6] if each non-ground functional subterm of the atom contains all variables of the atom. Finally, Theorem 1 by de Nivelle [26] says that, for $\theta$ 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.

Now consider arbitrary rules $\tau = \beta \to H$ and $\tau' = A' \wedge \beta' \to H'$ and an MGU $\theta$ of $H$ and $A'$ satisfying the preconditions of the SkDR inference rule. Atom $H$ thus contains a Skolem symbol, and rule $\tau$ 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 $\tau'$, and it is clearly weakly covering. By Theorem 1 of de Nivelle [26], atom $\theta(A')$ is weakly covering and has variable depth at most one; hence, 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 $\theta$ 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 Theorem 1 of de Nivelle [26], atom $\theta(H) = \theta(A')$ is weakly covering and has variable depth at most one, which can be only if $\theta$ 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. □

Theorem 5.17. *Each* SkDR*-rewriting of each finite set of GTGDs $\Sigma$ is a Datalog rewriting of $\Sigma$. Moreover,* SkDR$(\Sigma)$ *can be computed in time $O(b^{r^d \cdot (e+w_b+c)^{da}})$ for $r$ the number of relations in $\Sigma$, $a$ the maximum relation arity in $\Sigma$, $e$ the number of existential quantifiers in $\Sigma$, $w_b = \mathsf{bwidth}(\Sigma)$, $c = |\mathsf{consts}(\Sigma)|$, and some constants $b$ and $d$.*

Proof of Correctness. Fix an arbitrary finite set of GTGDs $\Sigma$, fix an arbitrary $\Sigma' \subseteq \mathsf{SkDR}(\Sigma)$ that contains SkDR$(\Sigma)$ up to redundancy, and let $\Sigma''$ be the set of Skolem-free Datalog rules of $\Sigma'$. It is easy to see that SkDR$(\Sigma)$ is a logical consequence of the Skolemization of $\Sigma$, so $\Sigma'$ and $\Sigma''$ are logical consequences of the Skolemization of $\Sigma$ as well. Moreover, $\Sigma''$ contains each full GTGD of $\Sigma$ up to redundancy, so each full GTGD of $\Sigma$ is logically entailed by $\Sigma''$. We next consider an arbitrary base instance $I$ and a one-pass tree-like chase sequence for $I$ and $\Sigma$, and we show the following property:

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

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

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 $T_i, T_{i+1}, \dots, T_{j-1}, T_j$ at vertex $v$, and assume that the claim holds for all shorter loops. 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 $\gamma$. 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 $\Sigma$-guarded by the instantiated head of $\tau$, 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 $\tau$, let $\nu$ 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 $\nu^-$ is well-defined, and we define $\nu^-(U)$ for $U$ a set of facts in the obvious way. By Proposition 4.8 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 $\Sigma$-guarded by $T_i(v)$ and would thus be propagated back to $v$. We now show that the following property holds for each $k$ with $i < k \leq j-1$:

($\lozenge$) for each fact $G \in T_k(v') \setminus S$, there exist a rule $\beta \to H \in \Sigma'$ and a substitution $\sigma$ such that $\beta$ 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 $\Sigma'$ contains the rules obtained by Skolemizing GTGD $\tau$. 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 $\Sigma$ derives $G$ from $T_k(v')$. Set $\Sigma'$ 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 $\sigma'$ 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 $\sigma'$ 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 $\Sigma'$ 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 $\Sigma'$ 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 $1 \leq \ell \leq n$. Towards this goal, we shall define $n$ rules $\beta''_\ell \to H'_\ell$, substitutions $\sigma'_\ell$, and sets of atoms $W = W_0 \supseteq \cdots \supseteq W_n$ for $\ell$ with $0 \leq \ell \leq n$ 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 $0 \leq \ell < n$, so $\beta''_\ell \to H'_\ell$, $\sigma'_\ell$, and $W_\ell$ 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 $A'_\ell$ 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 $A'_\ell$ 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 $\sigma'_\ell(A'_\ell) = \sigma'(B') \in T_k(v') \setminus S$. Thus, by property $(\lozenge)$, there exist a rule $\beta_\ell \to H_\ell \in \Sigma'$ and a substitution $\sigma_\ell$ such that $\beta_\ell$ is Skolem-free, $\sigma_\ell(\beta_\ell) \subseteq \nu(T_i(v))$, and $\sigma_\ell(H_\ell) = \sigma'(B')$; the last observation ensures that $H_\ell$ contains a Skolem symbol. Moreover, there exists an MGU $\theta_\ell$ of $H_\ell$ and $A'_\ell$, so the SkDR inference rule is applicable to $\beta_\ell \to H_\ell$ and $A'_\ell \wedge \beta'_\ell \to H'_\ell$, and $\Sigma'$ 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 $\theta$; 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. Now 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, Definition 5.1 ensures that 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 where $\sigma'_{\ell+1}(x) = \zeta_\ell(\mu_{\ell+1}(x))$ for each variable $x$ in $\beta''_{\ell+1} \to H'_{\ell+1}$. Then, property $(*)$ clearly holds for $\beta''_{\ell+1} \to H'_{\ell+1}$, $\sigma'_{\ell+1}$, and $W_{\ell+1}$, as required. □

Proof of Complexity. Fix $\Sigma$, $r$, $w_b$, $e$, $c$, and $a$ as stated in the theorem. Skolemizing a GTGD $\forall\vec{x}[\beta \to \exists\vec{y}\,\eta]$ 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 $\Sigma'$ 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 $\ell$ body atoms of one rule with the head atom of the other rule, so the unifier $\theta$ may need to be computed at most $\wp^2 \cdot \ell \leq \wp^2 \cdot 2^\ell = 2^{5\ell} = 32^\ell$ times. Finally, unification of atoms requires time that is linear in $a$, and all other steps require linear time too. □

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. Note that no GTGD derived by ExbDR in the proof of Propositions 5.18 is redundant, whereas the rules derived by SkDR in the proof of Proposition 5.19 become redundant once the rule (52) is defined.

Proposition 5.18. *There exists a family $\{\Sigma_n\}_{n\in\mathbb{N}}$ of finite sets of GTGDs such that $|\mathsf{ExbDR}(\Sigma_n)|$ is $\Omega(2^n)$ times larger than $|\mathsf{SkDR}(\Sigma_n)|$ for each $n > 1$.*

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

$$A(x) \to \exists\vec{y}\, B_1(x, y_1) \wedge \cdots \wedge B_n(x, y_n) \tag{43}$$

$$B_i(x_1, x_2) \wedge C_i(x_1) \to D_i(x_1, x_2) \text{ for } 1 \leq i \leq n \tag{44}$$

On $\Sigma_n$, ExbDR derives a GTGD of the form (45) for each subset $\{k_1, \ldots, k_m\} \subseteq \{1, \ldots, n\}$, and there are $2^n$ such TGDs. In contrast, the Skolemization of (43) consists of $n$ rules shown in equation (46), so SkDR derives just $n$ rules shown in equation (47).

$$A(x) \wedge \bigwedge_{i=1}^{m} C_{k_i}(x) \to \exists\vec{y} \bigwedge_{i=1}^{n} B_i(x, y_i) \wedge \bigwedge_{i=1}^{m} D_{k_i}(x, y_{k_i}) \tag{45}$$

$$A(x) \to B_i(x, f_i(x)) \text{ for } 1 \leq i \leq n \tag{46}$$

$$A(x) \wedge C_i(x) \rightarrow D_i(x, f_i(x)) \text{ for } 1 \leq i \leq n \qquad \square \tag{47}$$

Proposition 5.19. *There exists a family $\{\Sigma_n\}_{n\in\mathbb{N}}$ of finite sets of GTGDs such that $|\mathsf{SkDR}(\Sigma_n)|$ is $\Omega(2^n)$ times larger than $|\mathsf{ExbDR}(\Sigma_n)|$ for each $n > 1$*

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

$$A(x) \rightarrow \exists y\, B_1(x, y) \wedge \cdots \wedge B_n(x, y) \tag{48}$$

$$B_1(x_1, x_2) \wedge \cdots \wedge B_n(x_1, x_2) \rightarrow C(x_1) \tag{49}$$

On $\Sigma_n$, ExbDR derives just GTGD (50) in one step. In contrast, the Skolemization of (48) consists of $n$ rules of the form (51) for each $1 \leq i \leq n$. Thus, SkDR combines these with (49) to derive $2^n - 1$ rules of the form (52), one for each subset $\{k_1, \ldots, k_m\} \subsetneq \{1, \ldots, n\}$.

$$A(x) \rightarrow C(x) \tag{50}$$

$$A(x) \rightarrow B_i(x, f(x)) \tag{51}$$

$$A(x) \wedge B_{k_1}(x, f(x)) \wedge \cdots \wedge B_{k_m}(x, f(x)) \rightarrow C(x) \qquad \square \tag{52}$$

### 5.4 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 5.19. 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 5.20.** *The* Hyperresolution Rewriting *inference rule* HypDR *takes guarded rules*

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

*such that*

- *for each $i$ with $1 \leq i \leq n$, conjunction $\beta_i$ is Skolem-free and atom $H_i$ contains a Skolem symbol, and*
- *rule $\tau'$ is Skolem-free,*

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

$$\theta(\beta_1) \wedge \cdots \wedge \theta(\beta_n) \wedge \theta(\beta') \rightarrow \theta(H').$$

Example 5.21. 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 the set $\Sigma$ of GTGDs from Example 4.3.

First, HypDR derives (40) from (36) and (9), and it derives (14) from (10), (35), and (40). Rule (14) is derived just once, and without intermediate rules (39) and (41). In other words, the HypDR inference rule does not resolve the body atoms of a rule in every possible order. As Proposition 5.24 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 (42). 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 5.22 and Theorem 5.23 capture the properties of HypDR, and Proposition 5.24 compares it to SkDR.

Proposition 5.22. *Each application of the* HypDR *inference rule to rules $\tau_1, \ldots, \tau_n$ and $\tau'$ as in Definition 5.20 produces a guarded rule.*

Proof. Consider arbitrary rules $\tau_i = \beta_i \to H_i$ with $1 \leq i \leq n$ such that $\beta_i$ is Skolem-free and $H_i$ contains a Skolem symbol, a Skolem-free rule $\tau' = A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$, and an MGU $\theta$ of $H_1, \ldots, H_n$ and $A'_1, \cdots A'_n$ satisfying the preconditions of the HypDR inference rule. Rule $\tau_1$ 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 $\tau'$. Moreover, rule $\tau'$ 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 $\tau'$ 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 5.16. □

Theorem 5.23. *Each* HypDR*-rewriting of each finite set of GTGDs $\Sigma$ is a Datalog rewriting of $\Sigma$. Moreover,* HypDR($\Sigma$) *can be computed in time $O(b^{r^d \cdot (e+w_b+c)^{da}})$ for $r$ the number of relations in $\Sigma$, $a$ the maximum relation arity in $\Sigma$, $e$ the number of existential quantifiers in $\Sigma$, $w_b = \mathsf{bwidth}(\Sigma)$, $c = |\mathsf{consts}(\Sigma)|$, and some constants $b$ and $d$.*

Proof of Correctness. The correctness proof for HypDR is almost identical to the correctness proof in Theorem 5.17, so we outline just the difference. In particular, we wish to prove properties (⧫) and (◊) exactly as stated in Theorem 5.17 using the same proof structure. In the proof of property (◊), we establish existence of a Skolem-free rule $\beta'' \to H' \in \Sigma'$ and a substitution $\sigma'$ 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 5.17 is that we "resolve away" all relevant body atoms of $\beta''$ in one step. To this end, let $A'_1, \ldots, A'_n$ be precisely the atoms of $\beta''$ 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 $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$, and $\sigma'(\beta') \subseteq S$ clearly holds. By property (◊), for each $1 \leq \ell \leq n$, there exist a rule $\beta_\ell \to H_\ell \in \Sigma'$ and substitution $\sigma_\ell$ such that $\beta_\ell$ is Skolem-free and $\sigma_\ell(H_\ell) = \sigma'(A'_\ell)$; the last observation ensures that $H_\ell$ contains a Skolem symbol. Finally, there exists an MGU $\theta$ 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 $\beta_1 \to H_1, \ldots, \beta_n \to H_n$ and $A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \to H'$, so set $\Sigma'$ contains rule $\theta(\beta_1) \wedge \cdots \wedge \theta(\beta_n) \wedge \theta(\beta') \to \theta(H')$ up to redundancy. Since no premises share variables, substitution $\sigma_1 \cup \cdots \cup \sigma_n \cup \sigma'$ is correctly defined, so let $\zeta$ be the composition of $\sigma_1 \cup \cdots \cup \sigma_n \cup \sigma'$ and $\theta$. 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 $\theta(\beta_1) \wedge \cdots \wedge \theta(\beta_n) \wedge \theta(\beta') \to \theta(H')$ is not a syntactic tautology so, by Definition 5.1, there exist a rule $\beta \to H \in \Sigma'$ and substitution $\mu$ such that $\mu(\beta) \subseteq \theta(\beta_1) \cup \cdots \cup \theta(\beta_n) \cup \theta(\beta')$ and $\mu(H) = \theta(H')$. Let $\sigma$ be the substitution defined on each variable $x$ in $\beta \to H$ such that $\sigma(x) = \zeta(\mu(x))$. Then, $\sigma(\beta) \subseteq S$ and $\sigma(H) = \sigma'(H') = \nu(G)$, as required for property (◊). □

Proof of Complexity. Fix $\Sigma$, $r$, $w_b$, $e$, $c$, and $a$ as stated in the theorem. In the same way as in the complexity proof of Theorem 5.17, 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 $\tau'$ and then, for each of the at most $\ell$ body atoms in $\tau'$, we select one of the $\wp$ rules that play the role of rules $\tau_i$. 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 $\theta$ at most $(2^{2\ell})^{\ell+1} = 2^{2\ell^2+2\ell} \leq 2^{3\ell^2}$ times. Finally, the times needed for unification and all other steps can be bounded analogously as in the complexity proof of Theorem 5.17. □

Regarding the relative efficiency of HypDR and ExbDR, Proposition 5.18 equally applies to HypDR since this algorithm performs the same inferences as SkDR on the GTGDs considered in the proof. For the converse relation, we conjecture that each HypDR inference corresponds to an ExbDR inference, but proving this seems nontrivial: the HypDR inference rule is applied to $n$ rules $\tau_1, \ldots, \tau_n$ with Skolem symbols, while an analogous ExbDR inference is applied to just one non-full GTGD $\tau$, and the relation between these rules/GTGDs is not obvious.

Regarding the relative efficiency of HypDR and SkDR, one can see that SkDR derives exponentially more rules on the GTGDs from the proof of Proposition 5.19, but only if redundant rules are not eliminated. Proposition 5.24 proves a slightly stronger result: HypDR can be more efficient even with redundancy elimination. Furthermore, Proposition 5.25 shows that SkDR always performs as many inferences as HypDR when redundant rules are not eliminated. With redundancy elimination, we found analyzing this relationship difficult because the outputs of the two algorithms then depend on when and how redundant clauses are discarded.

Proposition 5.24. *There exists a family $\{\Sigma_n\}_{n\in\mathbb{N}}$ of finite sets of GTGDs such that $|\mathsf{SkDR}(\Sigma_n)|$ is $\Omega(2^n)$ times larger than $|\mathsf{HypDR}(\Sigma)|$ for each $n > 1$.*

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

$$A(x) \rightarrow \exists y\, B(x, y) \tag{53}$$

$$B(x_1, x_2) \wedge C_i(x_1) \rightarrow D_i(x_1, x_2) \text{ for } 1 \leq i \leq n \tag{54}$$

$$D_1(x_1, x_2) \wedge \cdots \wedge D_n(x_1, x_2) \rightarrow E(x_1) \tag{55}$$

Skolemizing (53) produces (56). Thus, SkDR combines (56) with each (54) to derive each (57), and it uses (57) and (55) to derive $2^n - 1$ rules of the form (58) for each set of indexes $I$ satisfying $\emptyset \subsetneq I \subseteq \{1, \ldots, n\}$; note that none of these rules are redundant.

$$A(x) \rightarrow B(x, f(x)) \tag{56}$$

$$A(x) \wedge C_i(x) \rightarrow D_i(x, f(x)) \text{ for } 1 \leq i \leq n \tag{57}$$

$$A(x) \wedge \bigwedge_{i\in I} C_i(x) \wedge \bigwedge_{j\in\{1,\ldots,n\}\setminus I} D_j(x, f(x)) \rightarrow E(x) \tag{58}$$

In contrast, HypDR derives each (57) just like SkDR, and it combines in one step (55) and all (57) to derive (58) for $I = \{1, \ldots, n\}$. □

Proposition 5.25. *For each finite set of GTGDs $\Sigma$, it is the case that $\mathsf{HypDR}(\Sigma) \subseteq \mathsf{SkDR}(\Sigma)$.*

Proof. The proof is by a simple induction on the construction of $\mathsf{HypDR}(\Sigma)$. The base case holds trivially. For the induction step, consider an arbitrary subset $\Sigma' \subseteq \mathsf{HypDR}(\Sigma)$ containing rules $\tau_i = \beta_i \rightarrow H_i$ for $1 \leq i \leq n$ and rule $\tau' = A'_1 \wedge \cdots \wedge A'_n \wedge \beta' \rightarrow H'$ such that the preconditions of the HypDR inference rule from Definition 5.20 are satisfied for MGU $\theta$, and the resulting rule $\tau = \theta(\beta_1) \wedge \cdots \wedge \theta(\beta_n) \wedge \theta(\beta') \rightarrow \theta(H')$ is not contained in $\Sigma'$ up to variable renaming. Since $\theta(\beta')$ is Skolem-free and the head of each $\tau_i$ contains a Skolem symbol, at least one atom among $A'_1, \ldots, A'_n$ must contain all variables of $\tau'$; without loss of generality, we can assume this to be atom $A'_1$. Moreover, each $\theta(A'_i)$ contains a Skolem symbol. But then, it should be clear that we can derive $\theta$ by successively applying the SkDR rule to $\tau_1$ and $\tau'$, $\tau_2$ and the result of the previous step, and so on. □

## 6 Implementation and Optimizations

In this section, we discuss numerous issues that have to be addressed to make the computation of a rewriting feasible for nontrivial sets of input GTGDs.

**Algorithm 1** Computing an Inf-rewriting of a finite set of GTGDs $\Sigma$

1: $\mathcal{W} = \emptyset$
2: $\mathcal{U}$ = the head-normal form or the Skolemization of $\Sigma$
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 $\tau$ and a subset of $\mathcal{W}$ and transforming the result into head-normal form
7: **for each** $\tau' \in \mathcal{E}$ **do**
8: **if** $\tau'$ is not contained in $\mathcal{W} \cup \mathcal{U}$ up to redundancy **then**
9: Remove from $\mathcal{W}$ and $\mathcal{U}$ each $\tau''$ subsumed by $\tau'$
10: $\mathcal{U} = \mathcal{U} \cup \{\tau'\}$
11: **return** $\{\tau \in \mathcal{W} \mid \tau$ is a Skolem-free Datalog rule$\}$

*Computing a Small* Inf*-Rewriting in Practice.* Definition 5.3 does not specify how to compute the set $\Sigma'$, and redundancy elimination makes this question nontrivial. When Inf derives a TGD/rule $\tau$, we can apply subsumption in two ways. First, we can discard $\tau$ if $\tau$ is subsumed by a previously derived TGD/rule; this is known as *forward subsumption*. Second, if $\tau$ is not discarded, we can discard each previously derived TGD/rule that is subsumed by $\tau$; 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* [44] that is widely 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 still need to be processed. Set $\mathcal{W}$ is initially empty (line 1), and set $\mathcal{U}$ is initialized to the head-normal form of $\Sigma$ if Inf manipulates TGDs, or to the Skolemization of $\Sigma$ if Inf manipulates rules. Next, the algorithm processes each $\tau \in \mathcal{U}$ iteratively until set $\mathcal{U}$ becomes empty (lines 3–10). It is generally beneficial to process shorter TGDs/rules first as that improves chances of redundancy elimination. After moving $\tau$ to $\mathcal{W}$ (line 5), the algorithm applies Inf to $\tau$ 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 $\tau'$ is not discarded, the algorithm applies backward subsumption to $\tau'$, $\mathcal{W}$, and $\mathcal{U}$ (line 9) and adds $\tau'$ 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 termination, set $\mathcal{W}$ satisfies the condition on $\Sigma'$ from Definition 5.3.

*Checking Subsumption.* Checking whether TGD/rule $\tau_1$ subsumes $\tau_2$ is NP-complete [39], and the main difficulty is in matching the variables of $\tau_1$ to the variables of $\tau_2$. 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 $i^{th}$ 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 an analogous 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) $\Sigma$.

*Subsumption Indexing.* Sets $\mathcal{W}$ and $\mathcal{U}$ can be large, so we use a variant of *feature vector indexing* [52] to retrieve subsumption candidates in $\mathcal{W} \cup \mathcal{U}$. For brevity, we discuss next only TGDs, but rules can be handled analogously. A TGD $\tau_1$ can subsume TGD $\tau_2$ only if the set of relations occurring in the body of $\tau_1$ (respectively the head of $\tau_2$) is a subset of the set of relations occurring in the body of $\tau_2$ (respectively the head of $\tau_1$). Thus, retrieving subsumption candidates can be reduced to following problem: given a domain set $D$, a set $N$ of subsets of $D$, a subset $S \subseteq D$, and $\bowtie \in \{\subseteq, \supseteq\}$, retrieve each $S' \in N$ satisfying $S' \bowtie S$. The set-trie data structure [51] 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 $\Sigma$, and we represent each TGD $\tau$ as a *feature vector* $F_\tau$ of these symbols corresponding to the body and head of $\tau$. Moreover, we combine the subset and superset retrieval algorithms in the obvious way. For example, when searching for a TGD $\tau' \in \mathcal{W} \cup \mathcal{U}$ that subsumes a given TGD $\tau$, 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 $\Sigma$ of input TGDs, and moreover we order each $R^b$ before all $R^h$.

*Relation Clustering.* Since our subsumption indexes can get very large, index traversal can become a significant 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 $F_\tau$ associated with each TGD $\tau$ consists of all clusters $C^b$ and $C^h$ containing a relation occurring in the body and head, respectively, of $\tau$. We also adapt the trie traversal algorithms in the obvious way. 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 $\tau$. 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* [53]: 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 $\tau$, we consider each body and head atom $A$ of $\tau$, we convert $A$ into the corresponding sequence that we use to query the relevant trie for all candidates participating in an inference with $\tau$ on $A$.

*Cheap Lookahead Optimization.* Consider an application of the ExbDR inference rule to GTGDs $\tau$ and $\tau'$ as in Definition 5.8, producing a GTGD $\tau''$ where $\mathsf{vars}(\theta(H')) \cap \vec{y} \neq \emptyset$ and the relation of $H'$ does not occur in the body of a GTGD in $\Sigma$. In each one-pass chase sequence for $\Sigma$, no GTGD of $\Sigma$ can be applied to a fact obtained by instantiating $\theta(H')$, so deriving this fact is redundant. Consequently, we can drop such $\tau''$ as soon as we derive it in line 6. Analogously, when the SkDR inference rule is applied to rules $\tau$ and $\tau'$ as in Definition 5.14, we can drop the resulting rule whenever $\theta(H')$ is not full and $\theta(H')$ contains a relation not occurring in the body of a GTGD in $\Sigma$.

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 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 setup and results are available online [19].

### 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 [48]. 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 [11], 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 [8], where class and properties correspond to unary and binary relations, respectively. 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 FullDR, 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 [36, 46, 47] 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 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

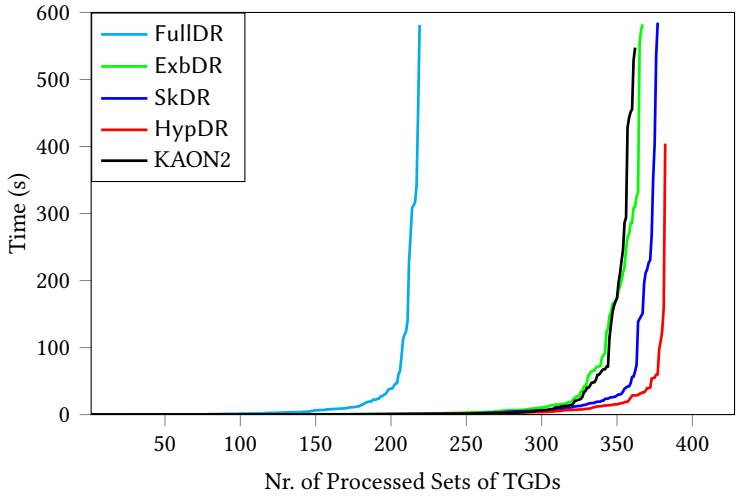
We computed the Datalog rewriting of GTGDs obtained from OWL ontologies using our four algorithms and KAON2. Figure 4 shows the number of inputs that each algorithm processed in a given time, provides information about the inputs and outputs of each system, and compares the performance among systems. The input size for FullDR and 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 input sizes. Each input GTGD contained at most seven body atoms. Out of 428 inputs, 220 inputs were processed by all five systems within the ten-minute limit. After excluding FullDR, 349 inputs were processed by our remaining three systems, and 334 inputs were processed by our three systems and KAON2. Finally, 32 inputs, consisting of between 20,270 and 221,648 GTGD, were not processed by any system.

*The Performance of* FullDR. As one can see in Figure 4, FullDR is a clear outlier: it could process considerably fewer inputs than all other algorithms, its running times were considerably higher, and the computed rewritings were considerably larger. The main obstacle in FullDR is indeed the need to consider a large number of substitutions $\theta$, as we outlined in Example 5.6. Nevertheless, the algorithm could still process around half of the inputs, some containing more than 10k+ GTGDs—that is, it could process not only "toy" inputs.

*The Remaining Algorithms.* After excluding FullDR, the remaining 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 5.12, 5.17, and 5.23 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 universally the best, but HypDR seems to offer superior 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 5.21. Specifically, HypDR derives rules with just one head atom, but it does not derive intermediate rules with function symbols in body atoms. The main source of overhead in HypDR seems to be the 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 reran ExbDR, SkDR, and HypDR with a slight 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




| | | FullDR | ExbDR | SkDR | HypDR | KAON2 |
|---|---|---|---|---|---|---|
| # of Processed Inputs | | 223 | 367 | 377 | 382 | 362 |
| Max. Processed Input Size | | 11,846 | 185,515 | 324,092 | 324,092 | N/A |
| Max. Output Size | | 229,597 | 196,594 | 124,846 | 124,846 | 61,964 |
| Max. Size Blowup | | 3670.83 | 8.95 | 8.85 | 8.85 | N/A |
| Max. Body Atoms in Output | | 9 | 7 | 6 | 6 | 4 |
| # Blowup $\geq 1.5$ | | 186 | 26 | 14 | 16 | N/A |
| Time (s) | Min. | 0.08 | 0.05 | 0.05 | 0.04 | 0.21 |
| | Max. | 601.34 | 582.18 | 584.79 | 404.34 | 547.53 |
| | Avg. | 316.71 | 23.23 | 14.34 | 6.38 | 18.66 |
| | Med. | 1.83 | 0.82 | 0.52 | 0.55 | 0.49 |

| | $time(Y)/time(X) \geq 10$ | | | | | $X$ and $Y$ both fail | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $X$ \ $Y$ | FullDR | ExbDR | SkDR | HypDR | KAON2 | FullDR | ExbDR | SkDR | HypDR | KAON2 |
| FullDR | | 2 | 0 | 0 | 0 | 205 | | | | |
| ExbDR | 172 | | 19 | 0 | 19 | 58 | 61 | | | |
| SkDR | 205 | 37 | | 0 | 26 | 51 | 33 | 51 | | |
| HypDR | 221 | 37 | 12 | | 31 | 46 | 35 | 43 | 46 | |
| KAON2 | 164 | 35 | 15 | 0 | | 66 | 37 | 47 | 46 | 66 |

Fig. 4. Results for TGDs Derived from Ontologies

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 is applied to many rules, so selecting inference candidates clearly becomes more difficult as the number of candidates increases.

*Impact of Structural Transformation.* KAON2 uses *structural transformation* [9] to simplify ontology axioms before translating them into GTGDs. For example, axiom $A \sqsubseteq \exists B.\exists C.D$ is transformed

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 |

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 ExbDR, SkDR, and HypDR 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 unclear how to systematically transform TGDs where heads and bodies consist of "flat" conjunctions. We leave this question for future work.

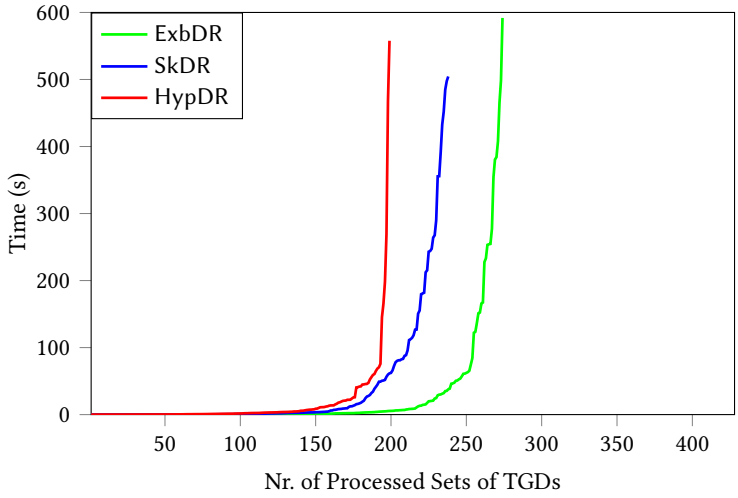
### 7.3 End-to-End Experiments

To validate our approach end-to-end, we selected ten ontologies on which ExbDR produced the largest rewritings; the IDs that can be used to identify these ontologies in the Oxford Ontology Library [48] are given in Table 2. For each ontology, we generated a large base instance using the WatDiv [3] generator, and we computed the fixpoint of the rewriting and the instance using the RDFox [54] 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 Datalog applications, but RDFox nevertheless computed all fixpoints in a few minutes. 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. Furthermore, all rewriting algorithms preserve the entailment of exactly the same facts, so using a different rewriting algorithm should not affect the fixpoint size. However, the rewritings produced by SkDR and HypDR are generally smaller than the rewritings produced by ExbDR, which typically allows for faster fixpoint computation. Consequently, 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. The FullDR algorithm performed very poorly on higher-arity inputs: increasing the arity increases the width of the input GTGDs, which additionally exacerbates the problems outlined in Example 5.6. Moreover, KAON2 supports relations of arity at most two and is thus not applicable to this experiment. Thus, we report the results for ExbDR, SkDR, and HypDR only. 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.




| | | 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 ≥ 1.5 | | 26 | 5 | 3 |
| Time (s) | Min. | 0.06 | 0.05 | 0.04 |
| | Max. | 591.82 | 504.49 | 557.75 |
| | Avg. | 26.70 | 38.39 | 17.05 |
| | 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 |

Fig. 5. Results for TGDs with Higher-Arity Relations

While HypDR was best-performing on GTGDs derived from ontologies, Figure 5 shows it to be worst-performing on higher-arity GTGDs: it could process only 199 inputs within the ten-minute timeout, while SkDR and ExbDR processed 238 and 274 inputs, respectively. This is due to the 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 number of GTGDs with relations of arity ten, so they can be seen as a kind of "stress test". Our algorithms were able to

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

In this paper, we presented four 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 so-called one-pass chase. We also presented the results of an empirical evaluation of our algorithms based on a new test set consisting of both real and synthetic GTGDs.

In our experiments, we compared our algorithms to KAON2, which is also based on rewriting. Fact entailment, however, can be solved using approaches other than rewriting (e.g., using forward chaining with blocking), so it would be interesting to compare the overall performance of fact entailment of such approaches and rewriting. For the class of guarded TGDs that we consider in this paper, we are not aware of any related approach that could be competitive with rewriting on base instances of nontrivial sizes. In our future work, however, we will aim to implement a forward chaining approach for GTGDs and then compare its performance to rewriting, analogously to how this was done recently for other classes of TGDs [2].

In addition, we plan to generalize our framework to wider classes of TGDs, such as frontier-guarded TGDs, as well as provide Datalog rewritings for conjunctive queries under certain answer semantics. Note that Datalog rewritings are known to exist for general conjunctive queries even for frontier-guarded TGDs [13, 14], but developing realistic algorithms is considerably more challenging than for fact entailment. Finally, we shall investigate the extension of our framework to disjunctive guarded TGDs. We presented an analog of the one-pass chase for disjunctive GTGDs, as well as an analog of the Skolem-based rewriting in an unpublished work [38]. It remains to be seen whether these ideas can be used to obtain practical algorithms for rewriting disjunctive guarded TGDs into disjunctive Datalog, analogously to how this was achieved for description logics [47].

## Acknowledgments

This work was funded by the EPSRC grants OASIS (EP/S032347/1), QUINTON (EP/T022124/1), UK FIRES (EP/S019111/1), AnaLOG (EP/P025943/1), and Concur (EP/V050869/1). For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

## References

[1] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus. 2018. Rewriting Guarded Existential Rules into Small Datalog Programs. In *Proc. of the 21st Int. Conf. on Database Theory (ICDT 2018) (LIPIcs, Vol. 98)*, Benny Kimelfeld and Yael Amsterdamer (Eds.). Schloss Dagstuhl—Leibniz-Zentrum für Informatik, Vienna, Austria, 4:1–4:24.

[2] Afnan G. Alhazmi, Tom Blount, and George Konstantinidis. 2022. ForBackBench: A Benchmark for Chasing vs. Query-Rewriting. *Proc. VLDB Endow.* 15, 8 (2022), 1519–1532.

[3] Günes Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. 2014. Diversified Stress Testing of RDF Data Management Systems. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2014) (LNCS, Vol. 8796)*, Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandecic, Paul Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble (Eds.). Springer, Riva del Garda, Italy, 197–212.

[4] Mario Alviano, Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. 2012. Magic-Sets for Datalog with Existential Quantifiers. In *Proc. of the 2nd Int. Workshop on Datalog in Academia and Industry (Datalog 2.0 2012) (LNCS, Vol. 7494)*, Pablo Barceló and Reinhard Pichler (Eds.). Springer, Vienna, Austria, 31–43.

[5] 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.

[6] Hajnal Andréka, Johan van Benthem, and István Németi. 1998. Modal Languages and Bounded Fragments of Predicate Logic. *J. Philos. Log.* 27 (1998), 217–274.

[7] F. Baader, S. Brandt, and C. Lutz. 2005. Pushing the $\mathcal{EL}$ Envelope. In *Proc. of the 19th Int. Joint Conference on Artificial Intelligence (IJCAI 2005)*, L. Pack Kaelbling and A. Saffiotti (Eds.). Morgan Kaufmann Publishers, Edinburgh, UK, 364–369.

[8] 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.

[9] M. Baaz, U. Egly, and A. Leitsch. 2001. Normal Form Transformations. In *Handbook of Automated Reasoning*, John Alan Robinson and Andrei Voronkov (Eds.). North-Holland, Amsterdam, Chapter 5, 273–333.

[10] Leo Bachmair and Harald Ganzinger. 2001. Resolution Theorem Proving. In *Handbook of Automated Reasoning*, John Alan Robinson and Andrei Voronkov (Eds.). North-Holland, Amsterdam, Chapter 2, 19–99.

[11] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Swan Rocher, and Clément Sipieter. 2015. Graal: A Toolkit for Query Answering with Existential Rules. In *Proc. of the 9th Int. Symposium on Rule Technologies: Foundations, Tools, and Applications (RuleML 2015) (LNCS, Vol. 9202)*, Nick Bassiliades, Georg Gottlob, Fariba Sadri, Adrian Paschke, and Dumitru Roman (Eds.). Springer, Berlin, Germany, 328–344.

[12] Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. 2011. Walking the Complexity Lines for Generalized Guarded Existential Rules. In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011)*, Toby Walsh (Ed.). AAAI Press, Barcelona, Spain, 712–717.

[13] Vince Bárány, Michael Benedikt, and Balder ten Cate. 2013. Rewriting Guarded Negation Queries. In *Proc. of the 38th Int. Symposium on Mathematical Foundations of Computer Science (MFCS 2013) (LNCS, Vol. 8087)*, Krishnendu Chatterjee and Jirí Sgall (Eds.). Springer, Klosterneuburg, Austria, 98–110.

[14] Vince Bárány, Michael Benedikt, and Balder ten Cate. 2018. Some Model Theory of Guarded Negation. *J. Symb. Log.* 83, 4 (2018), 1307–1344.

[15] Pablo Barceló, Gerald Berger, Carsten Lutz, and Andreas Pieris. 2018. First-Order Rewritability of Frontier-Guarded Ontology-Mediated Queries. In *Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI 2018)*, Jérôme Lang (Ed.). ijcai.org, Stockholm, Sweden, 1707–1713.

[16] 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.

[17] Michael Benedikt, Maxime Buron, Stefano Germano, Kevin Kappelmann, and Boris Motik. 2022. Datalog Rewriting for Guarded TGDs. In *Proc. of the 4th Int. Workshop on the Resurgence of Datalog in Academia and Industry (Datalog 2.0 2022) (CEUR Workshop Proceedings, Vol. 3203)*, Mario Alviano and Andreas Pieris (Eds.). CEUR-WS.org, Genova-Nervi, Italy, 104–113.

[18] Michael Benedikt, Maxime Buron, Stefano Germano, Kevin Kappelmann, and Boris Motik. 2022. Rewriting the Infinite Chase. *Proc. VLDB Endow.* 15, 11 (2022), 3045–3057.

[19] Michael Benedikt, Maxime Buron, Stefano Germano, Kevin Kappelmann, and Boris Motik. 2024. *Experiments on Guarded Saturation.* Zenodo. https://doi.org/10.5281/zenodo.13742630

[20] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the Chase. In *Proc. of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2017)*, Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts (Eds.). ACM, Chicago, IL, USA, 37–52.

[21] Andrea Calì, Georg Gottlob, and Michael Kifer. 2013. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.* 48 (2013), 115–174.

[22] Andrea Calì, Domenico Lembo, and Riccardo Rosati. 2003. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, Georg Gottlob and Toby Walsh (Eds.). Morgan Kaufmann, Acapulco, Mexico, 16–21.

[23] 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.

[24] 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.

[25] David Carral, Larry González, and Patrick Koopmann. 2019. From Horn-$\mathcal{SRIQ}$ to Datalog: A Data-Independent Transformation That Preserves Assertion Entailment. In *Proc. of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, Pascal Van Hentenryck and Zhi-Hua Zhou (Eds.). AAAI Press, Honolulu, HI, USA, 2736–2743.

[26] Hans de Nivelle. 1998. A Resolution Decision Procedure for the Guarded Fragment. In *Proc. of the 15th Int. Conf. on Automated Deduction (CADE 1998) (LNCS, Vol. 1421)*, Claude Kirchner and Hélène Kirchner (Eds.). Springer, Lindau, Germany, 191–204.

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

[28] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. 2005. Data exchange: semantics and query answering. *Theor. Comput. Sci.* 336, 1 (2005), 89–124.

[29] Mohamed Gaha, Arnaud Zinflou, Christian Langheit, Alexandre Bouffard, Mathieu Viau, and Luc Vouligny. 2013. An Ontology-Based Reasoning Approach for Electric Power Utilities. In *Proc. of the 7th Int. Conf. on Web Reasoning and Rule Systems (RR 2013) (LNCS, Vol. 7994)*, Wolfgang Faber and Domenico Lembo (Eds.). Springer, Mannheim, Germany, 95–108.

[30] Harald Ganzinger and Hans de Nivelle. 1999. A Superposition Decision Procedure for the Guarded Fragment with Equality. In *Proc. of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS 1999)*, Giuseppe Longo (Ed.). IEEE Computer Society, Trento, Italy, 295–303.

[31] Georg Gottlob, Sebastian Rudolph, and Mantas Simkus. 2014. Expressiveness of guarded existential rule languages. In *Proc. of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2014)*, Richard Hull and Martin Grohe (Eds.). ACM, Snowbird, UT, USA, 27–38.

[32] Alon Y. Halevy. 2001. Answering Queries Using Views: A Survey. *VLDB J.* 10, 4 (2001), 270–294.

[33] Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. 2006. Data Integration: The Teenage Years. In *Proc. of the 32nd Int. Conf. on Very Large Data Bases (VLDB 2006)*, Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim (Eds.). ACM, Seoul, Korea, 9–16.

[34] 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

[35] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. 2004. Reducing $\mathcal{SHIQ}^-$-Description Logic to Disjunctive Datalog Programs. In *Proc. of the 9th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2004)*, Didier Dubois, Christopher A. Welty, and Mary-Anne Williams (Eds.). AAAI Press, Whistler, BC, Canada, 152–162.

[36] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. 2007. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *J. Autom. Reason.* 39, 3 (2007), 351–384.

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

[38] Kevin Kappelmann. 2019. Decision Procedures for Guarded Logics. *CoRR* abs/1911.03679 (2019), 92 pages.

[39] Deepak Kapur and Paliath Narendran. 1986. NP-Completeness of the Set Unification and Matching Problems. In *Proc. of the 8th Int. Conf. on Automated Deduction (CADE 1986) (LNCS, Vol. 230)*, Jörg H. Siekmann (Ed.). Springer, Oxford, UK, 489–495.

[40] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. 2014. The Incredible ELK: From Polynomial Procedures to Efficient Reasoning with $\mathcal{EL}$ Ontologies. *J. Autom. Reason.* 53, 1 (2014), 1–61.

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

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

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

[44] William McCune and Larry Wos. 1997. Otter—The CADE-13 Competition Incarnations. *J. Autom. Reason.* 18, 2 (1997), 211–220.

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

[46] 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

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

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

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

[50] John Alan Robinson. 1965. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM* 12, 1 (1965), 23–41.

[51] Iztok Savnik. 2013. Index Data Structure for Fast Subset and Superset Queries. In *Proc. of the 5th Cross-Domain Conf. on Availability, Reliability, and Security in Information Systems and HCI (CD-ARES 2013) (LNCS, Vol. 8127)*, Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar R. Weippl, and Lida Xu (Eds.). Springer, Regensburg, Germany, 134–148.

[52] Stephan Schulz. 2013. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In *Automated Reasoning and Mathematics—Essays in Memory of William W. McCune (LNCS, Vol. 7788)*, Maria Paola Bonacina and Mark E. Stickel (Eds.). Springer, Berlin, Heidelberg, 45–67.

[53] Mark E. Stickel. 1989. *The Path-Indexing Method for Indexing Terms.* Technical Report. SRI. Retrieved July 27, 2022 from https://apps.dtic.mil/sti/citations/ADA460990

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

[55] Moshe Y. Vardi. 1996. Why is Modal Logic So Robustly Decidable?. In *Proc. of a DIMACS Workshop on Descriptive Complexity and Finite Models (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 31)*, Neil Immerman and Phokion G. Kolaitis (Eds.). DIMACS/AMS, Princeton, NJ, USA, 149–183.

[56] 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 *Proc. of the 28th Int. IEEE Conference on Data Engineering (ICDE 2012)*, Anastasios Kementsietsidis and Marcos Antonio Vaz Salles (Eds.). IEEE Computer Society, Washington, DC, USA, 1309–1312.

[57] Zhe Wang, Peng Xiao, Kewen Wang, Zhiqiang Zhuang, and Hai Wan. 2020. Query Answering for Existential Rules via Efficient Datalog Rewriting. In *Proc. of the 29th Int. Joint Conf. on Artificial Intelligence (IJCAI 2020)*, Christian Bessiere (Ed.). ijcai.org, Yokohama, Japan (Virtual event), 1933–1939.

[58] Sen Zheng and Renate A. Schmidt. 2020. Deciding the Loosely Guarded Fragment and Querying Its Horn Fragment Using Resolution. In *Proc. of the 34th AAAI Conf. on Artificial Intelligence (AAAI 2020)*, Vincent Conitzer and Fei Sha (Eds.). AAAI Press, New York, NY, USA, 3080–3087.

Received 26 January 2023; revised 27 February 2024; accepted 9 September 2024