Distribution Constraints: The Chase for Distributed Data

Gaetano Geck
TU Dortmund University, Germany
gaetano.geck@tu-dortmund.de

Frank Neven
Hasselt University and transnational University of Limburg, Belgium
frank.neven@uhasselt.be

Thomas Schwentick
TU Dortmund University, Germany
thomas.schwentick@tu-dortmund.de

Abstract

This paper introduces a declarative framework to specify and reason about distributions of data over computing nodes in a distributed setting. More specifically, it proposes distribution constraints which are tuple and equality generating dependencies (tgds and egds) extended with node variables ranging over computing nodes. In particular, they can express co-partitioning constraints and constraints about range-based data distributions by using comparison atoms. The main technical contribution is the study of the implication problem of distribution constraints. While implication is undecidable in general, relevant fragments of so-called data-full constraints are exhibited for which the corresponding implication problems are complete for EXPTIME, PSPACE and NP. These results yield bounds on deciding parallel-correctness for conjunctive queries in the presence of distribution constraints.

2012 ACM Subject Classification Theory of computation → Database constraints theory; Theory of computation → Logic and databases; Information systems → Parallel and distributed DBMSs

Keywords and phrases tuple-generating dependencies, chase, conjunctive queries, distributed evaluation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.13

Related Version A full version of this paper is available on arXiv [27], `https://arxiv.org/abs/2003.00965`.

Acknowledgements We thank Michael Benedikt, Bas Ketsman, Andreas Pieris, Phokion Kolaitis, Christopher Spinrath, Brecht Vandevoort, and Thomas Zeume for helpful discussions on various aspects of this work.

1 Introduction

Distributed storage and processing of data has been used and studied since the 1970s and became more and more important in the recent past. One of the most fundamental questions in distributed data management is the following: *how should data be replicated and partitioned over the set of computing nodes?* It is paramount to answer this question well as the placement of data determines the reliability of the system and is furthermore critical for its scalability including the performance of query processing.

On the one hand, despite the importance of this question and decades of research, the placement strategies remained rather simple for a long time: horizontal or vertical fragmentation of relations – or hybrid variants thereof [37]. These placement strategies often require a reshuffling of the data for each binary join in the processed query which are commonly based on a range or hash partitioning of the relevant attributes. Recently, however, more elaborated schemes of data placement like co-partitioning, single hypercubes (for multiway-joins) or multiple hypercubes (for skewed data) gained some attention [3, 12, 30, 39, 41, 45].

On the other hand, there is a long tradition in studying tuple- and equality-generating dependencies (tgds/egds) as a simple but versatile tool to describe relationships among relational data. The research on these dependencies focuses mainly on the implication[1] problem. More precisely, since the implication problem in general is undecidable, several fragments have been considered in an attempt to locate the boundaries of decidability and complexity. Commonly, these fragments are defined by syntactical restrictions on the sets of dependencies, like weak acyclicity, weak guardedness, stickiness, wardedness, ... [15–17, 23].

It seems desirable to connect these two strands of research. Being able to reason about the placement of data offers database management systems additional optimisation potential, for instance, when it comes to the placement of new data or when the cost of a query execution plan is estimated. In the latter case, a reshuffling phase, which often dominates the processing time, can sometimes be omitted completely because the query at hand is already parallel-correct[2] under the current distribution.

The goal of this paper is to make a first step towards a connection between existing partitioning schemes and well-known reasoning frameworks. With this intent, we introduce *distribution constraints* – a variant of tgds/egds that is specifically geared towards distributed data – and study its implication problem. In particular, we identify fragments of distribution constraints by the complexity of the associated implication problem. Although the implication problem is certainly not the only – and, admittedly, not the most innovative – problem related to reasoning about distributed data, it is yet a basic problem that is likely to have connections to other algorithmical questions centering around this topic (like *how to derive a new distribution for the next query, making use of the current distribution?*).

Contributions. We start by defining *distribution constraints* as tgds and egds with atoms of the form $R(x,y)@\kappa$, in which κ is understood as a node variable with the intended meaning that fact $R(x,y)$ is at node κ. To achieve decidability, we further require that distribution tgds are *data-full*, i.e., only node variables may be quantified existentially.

We demonstrate that distribution constraints can express several common distribution schemes, incorporating range and hash partitionings [37], co-partitionings [21, 25], hierarchical partitionings (as used in Google's F1 [39, 41]), predicate-based reference partitionings [45], hypercube distributions [3, 12], and multi-round communication.

▶ **Example 1.** As an example, consider the following set of distribution tgds, describing a "derived horizontal" fragmentation [37] of relation `Msg` based on the `Range`-predicate and the message's sender id s:

$$\begin{aligned}&\texttt{Range}(\ell,u) \to \texttt{Range}(\ell,u)@\kappa,\\ &\texttt{Msg}(s,r) \to \texttt{Msg}(s,r)@\kappa,\\ &\texttt{Msg}(s,r)@\kappa, \texttt{Range}(\ell,u)@\lambda, \ell \leq s, s \leq u \to \texttt{Msg}(s,r)@\lambda\end{aligned}$$

The first two rules enforce that, for every `Range`- and every `Msg`-fact, there is a responsible node (indicated by the node variable κ). The third rule ensures that every `Msg`-fact can be found at every node whose `Range`-bounds match the sender id. We remark that the above set of constraints implies the following distribution tgd:

$$\texttt{Msg}(s_1,r), \texttt{Msg}(s_2,r), \texttt{Range}(\ell,u), \ell \leq s_1, s_1 \leq u, \ell \leq s_2, s_2 \leq u \to \texttt{Msg}(s_1,r)@\kappa, \texttt{Msg}(s_2,r)@\kappa,$$

[1] Does a dependency τ always hold if a set Σ of dependencies is satisfied, $\Sigma \models \tau$?
[2] Parallel correctness is a basic notion of distributed query evaluation [7], also addressed in Section 3.3.

which states that all pairs of messages with the same receiver can be found at a common node if their senders fall in the same range. In other words, if the above set of constraints is satisfied over a distributed instance, then so is the just mentioned dtgd.

On the technical level, we show that the implication problem is EXPTIME-complete for these constraints in general, and we identify classes of distribution constraints where the complexity drops to PSPACE or even NP and classes where this is not the case. These classes are determined by simple syntactic criteria based on the amount of data associated with node variables.

Since distribution constraints incorporate all *full* tgds (without existential quantification), EXPTIME-hardness of their implication problem readily follows from an early result by Chandra, Lewis and Makowsky [19]. However, the latter result relies on the use of relation atoms of arbitrarily high arity, while the EXPTIME-hardness results in this paper already hold for a fixed schema of maximum arity of 3 (or 2, w.r.t. data variables). The corresponding upper bounds are established by an adaptation of the standard chase procedure [23, 36].

The fragments studied here are defined depending on, first, the sizes of the node variables' *contexts* (the data variables occuring together with the node variable in some atom) and, second, on the distinction of data-collecting tgds and node-creating tgds (without/with existentially quantified node variable in the head). For a fixed integer b, a node variable has *bounded context* if its context size is at most b. Thanks to the obvious relationship between distribution constraints and standard constraints, the complexity results in this paper can also be viewed as results on fragments of standard tgds/egds.

Related work. There is a rich literature on restrictions of (sets of) tgds that yield a decidable (general and finite) implication problem [4, 36]. We discuss how our distributed constraints relate to classical constraints in Section 3. Restricting the use of existential variables in tgds is a common approach to define fragments of tgds that yield a decidable implication problem. Interestingly, the rather simple restriction to data-full dtgds studied here, is orthogonal to prominent examples like weak acyclicity, weak guardedness, stickiness and wardedness [15–17, 23].

Dependencies with arithmetic comparisons have been used in the context of Data Exchange [5, 43]. However, these papers mainly study full and weakly acyclic tgds and are thus orthogonal to our framework. There is further work on dependencies with stronger arithmetic constraints, e.g. [9, 11, 22, 33].

Declarative specifications for distributed data have also been studied before. Notable examples are Webdamlog and the already mentioned Data Exchange setting (which can be seen as a restricted form of distribution constraints with a global and a single local database). We refer to the book [8] for a relatively recent overview of Data Exchange.

Our notation $R(x)@\kappa$ for distributed atoms resembles that of Webdamlog, $R@\kappa(x)$, a dialect of datalog that was designed for distributed data management.[3] Besides implementing a system [2, 34] based on this dialect, the theoretical research on this language has mostly focussed on establishing a hierarchy among some of its fragments in terms of their expressiveness [3]. Neglecting the notational similarities, there seems to be no overlap between the research on Webdamlog – with its fixpoint evaluation mechanism (which even allows facts to vanish) – and the results on distribution constraints that we present in this paper.

[3] Annotated atoms have already been used before in Datalog dialects. For instance, in Dedalus [6], where they describe timestamps.

Particularly, Webdamlog seems to prohibit existential quantification of node variables and assumes, accordingly, that the number of nodes is explicitly fixed with the input. Distribution constraints, in contrast, *do allow* existential quantification of node variables, which affects the modeling capabilities and the complexity of the reasoning process.

Organisation of this paper. After providing the necessary preliminaries in Section 2, we formally define distribution constraints in Section 3, compare them with classical constraints, and give examples of their versatility. In Section 4, we define the implication problem and extend the standard chase to distribution constraints. In Section 5, we address the complexity of the implication problem and, finally, conclude in Section 6.

Due to space limitations, most proofs and some examples will be given in a full version, which we plan to publish on arXiv.

2 Preliminaries

In this section, we fix our notation for the basic concepts of this paper. Specific definitions for our framework are given in Section 3.

2.1 Databases and queries

Let dom and var be disjoint infinite sets of *data values* and *data variables*, respectively. For simplicity, we do not distinguish between different data types and assume that dom is linearly ordered. We denote data variables as usual with x, y, z, $\ldots$. A *schema* is a set $\mathcal{S}$ of relation symbols, where each relation symbol $R \in \mathcal{S}$ has some fixed arity $\mathsf{ar}(R)$. We write $\mathsf{ar}(\mathcal{S})$ for the *maximum* arity $\mathsf{ar}(R)$ of any $R \in \mathcal{S}$. A *relation atom over* $\mathcal{S}$ is of the form $R(t_1, \ldots, t_k)$ where R is a relation symbol of arity k and $t_1, \ldots, t_k \in \mathsf{dom} \cup \mathsf{var}$. A relation atom is a *fact* if $t_1, \ldots, t_k \in \mathsf{dom}$. A *comparison atom* is of the form $t < t'$ or $t \leq t'$ with $t, t' \in \mathsf{dom} \cup \mathsf{var}$. The set of data values occuring in a set $\mathcal{A}$ of (relational or comparison) atoms is denoted $\mathsf{adom}(\mathcal{A})$. Similarly, the set of variables in $\mathcal{A}$ is denoted $\mathsf{var}(\mathcal{A})$. Instances are finite sets of facts over a given schema $\mathcal{S}$.

A *valuation* for a set $\mathcal{A}$ of atoms is a mapping $V : \mathsf{var}(\mathcal{A}) \to \mathsf{dom}$. It *satisfies* $\mathcal{A}$ *on instance* I if $V(A) \in I$ holds for each relation atom $A \in \mathcal{A}$ and $V(t)\theta V(t')$ holds for each comparison atom $t\theta t'$ in $\mathcal{A}$. We often denote by V also the extension of V to dom defined by $V(a) = a$ for every $a \in \mathsf{dom}$.

A *conjunctive query* Q is of the form $S(x_1, \ldots, x_m) :- R_1(\boldsymbol{z}_1), \ldots, R_\ell(\boldsymbol{z}_\ell)$, where the *head* of the query, $\mathsf{head}_Q = S(x_1, \ldots, x_m)$, has a relation atom S not in $\mathcal{S}$ and its *body*, $\mathsf{body}_Q = \{R_1(\boldsymbol{z}_1), \ldots, R_\ell(\boldsymbol{z}_\ell)\}$, is a finite set of relation atoms over $\mathcal{S}$. In the following, all queries are assumed to be *safe*, that is, each variable in the head occurs at least once in some body atom. If V is a valuation that satisfies body_Q, we say that V *derives* fact $V(\mathsf{head}_Q)$. The *result* $Q(I)$ of query Q on instance I is the set of all derived facts.

2.2 Dependencies

A *tgd* σ is of the form $\mathcal{A}, \mathcal{C} \to \mathcal{A}'$, for sets $\mathcal{A}, \mathcal{A}'$ of relation atoms and a set $\mathcal{C}$ of comparison atoms with $\mathsf{var}(\mathcal{C}) \subseteq \mathsf{var}(\mathcal{A})$. Here, $\mathcal{A}'$ form its *head*, and $\mathcal{A}, \mathcal{C}$ its *body*, denoted $\mathsf{head}_\sigma = \mathcal{A}'$ and $\mathsf{body}_\sigma = \mathcal{A} \cup \mathcal{C}$, respectively. We refer to $\mathcal{A}$ by rbody_σ. The tgd is called *full* if $\mathsf{var}(\mathcal{A}') \subseteq \mathsf{var}(\mathcal{A})$. An instance I *satisfies* a tgd σ if, for every valuation V of body_σ that satisfies body_σ on I, there is an extension V' onto head_σ that satisfies head_σ on I.

An *egd* σ is of the form $\mathcal{A}, \mathcal{C} \to x = y$, for a set $\mathcal{A}$ of relation atoms and a set $\mathcal{C}$ of comparison atoms with $\mathsf{var}(\mathcal{C}) \cup \{x, y\} \subseteq \mathsf{var}(\mathcal{A})$. An instance I *satisfies* an egd σ if $V(x) = V(y)$ for every valuation V that satisfies body_σ on I, where body_σ is defined as for tgds.

Sets of dependencies are satisfied by an instance if each dependency in the set is satisfied. Satisfaction of a single dependency σ or a set Σ of dependencies by some instance I is denoted $I \models \sigma$ and $I \models \Sigma$, respectively.

A dependency τ is *implied* by a set Σ of dependencies, denoted $\Sigma \models \tau$, if $I \models \Sigma$ implies $I \models \tau$ for every instance I. For more precise statements, we can mention the actual domain in our notation. For example, we write $I \models_{\mathbb{N}} \tau$ if implication holds for all (finite) instances over $\mathbb{N}$.

We use the terms *dependencies* and *constraints* interchangeably.

2.3 Distributed Databases

We model a *network* of database servers as a finite set $\mathcal{N}$ of *nodes* and we denote its *size* by $|\mathcal{N}|$. We usually denote nodes by k and ℓ. A *distributed instance* $D = (G, \mathbf{I})$ consists of a *global instance* G and a family $\mathbf{I} = (I_k)_{k \in \mathcal{N}}$ of *local instances*, one for each node of $\mathcal{N}$, such that $\bigcup I_k \subseteq G$. We denote G by $\text{global}(D)$ and $(I_k)_{k \in \mathcal{N}}$ by $\text{local}(D)$.

We note that distributions allow redundant placement of facts, which is often desirable. Furthermore, it is not necessary to place all facts of the global instance on some node. A fact f is *skipped*[4] by D if $f \in \text{global}(D)$ but f does not occur in $\text{local}(D)$.

We write $f @_D k$ to denote that a fact f occurs at some node k, that is, $f \in I_k$. We drop D if it is clear from the context. We call $f @_D k$ a *distributed fact*. Sometimes we say that a set of facts *meet* in D when they all occur in the same local instance.

▶ **Example 2.** Consider a network $\mathcal{N} = \{1, 2\}$ of size 2 and a distributed instance $D = (G, \{I_1, I_2\})$ with $G = \{R(a, b), S(b), S(c), S(d)\}$, $I_1 = \{R(a, b), S(b)\}$ and $I_2 = \{S(b), S(c)\}$. Then fact $S(d)$ is skipped by D. The instance D can also be represented by the distributed facts $\{R(a, b), S(b), S(c), S(d), R(a, b)@1, S(b)@1, S(b)@2, S(c)@2\}$.

2.4 Parallel-correctness

Building on the computation model of massively parallel communication (MPC) [12], the naive evaluation of a conjunctive query Q over a distributed instance D evaluates Q separately for each local instance in $\text{local}(D)$. For $\text{local}(D) = (I_k)_{k \in \mathcal{N}}$, we write $Q_{\text{naive}}(D)$ for $\bigcup_{k \in \mathcal{N}} Q(I_k)$. Following [7], we say that a query Q is *parallel-correct* on D, if the naive evaluation produces the correct result, i.e., if $Q_{\text{naive}}(D) = Q(\text{global}(D))$.

3 Distribution constraints

We first introduce our framework for distribution constraints and afterwards give examples for its use.

[4] We note that allowing skipped facts makes the framework more flexible. They can be disallowed by simple distribution constraints, as discussed in Subsection 3.2.3.

3.1 Definition

Let nvar be an infinite set of *node variables* disjoint from dom and var. A *distributed atom* $A@\kappa$ consists of a relation atom A and a *node variable* κ in nvar. Recall that we refer to the variables of A as *data variables*. For a set of (distributed) atoms $\mathcal{A}$, we denote by $\mathsf{nvar}(\mathcal{A})$ the set of node variables occurring in atoms in $\mathcal{A}$. For a set $\mathcal{A}$ of relation atoms and a node variable κ, $\mathcal{A}@\kappa$ denotes the set $\{A@\kappa \mid A \in \mathcal{A}\}$.

Distribution tgds (dtgds) are defined just as tgds but they can additionally have distributed atoms in their body and their head. *Distribution egds (degds)* are defined just as egds but can have distributed atoms in their body. We do not allow node variables in comparison atoms (but we do allow them in the equality atom of a head in the case of degds). A degd $\mathcal{A}, \mathcal{C} \to A'$ is *node-identifying* if the equality atom A' refers to node variables only and *value-identifying* if, instead, A' refers to data variables only. We do *not* consider equality atoms where a node variable is identified with a data variable. We are particularly interested in *data-full* dtgds, for which the data variables in the head all occur in the body.

By $\mathcal{T}_{\text{all}}$ we denote the class of *all* dtgds and by $\mathcal{T}_{\text{df}}$ the class of data-full dtgds. By $\mathcal{E}_{\text{all}}$ we denote the class of *all* degds.

Satisfaction of dtgds and degds is defined in the obvious way with generalised valuations that may additionally map node variables to nodes. For a distributed atom $A' = A@\kappa$, we write $V(A') \in D$, if $V(A)@V(\kappa)$ is a distributed fact of D. For a relation atom A, we write $V(A) \in D$ if $V(A) \in \text{global}(D)$.

▶ **Example 3.** Given a schema $\mathcal{S}$ with binary relation symbols R and S, the following dtgd $\sigma = R(x, y), S(x, y) \to R(x, y)@\kappa, S(x, y)@\kappa$ is satisfied on a distributed instance D if, whenever global(D) contains two facts $R(a, b)$ and $S(a, b)$, for arbitrary data values $a, b \in \mathsf{dom}$, they meet in some local instance.

Below, in Section 3.2, we illustrate how distribution constraints can model global, local and global-to-local constraints.

▶ **Example 4.** The dtgd $E(x, y)@\kappa, E(y, z)@\kappa, E(z, x) \to E(z, x)@\kappa$ stipulates that every computing node has "complete" information w.r.t. open triangles on a binary relation E. That is, whenever a node contains two legs of a triangle, it also contains the closing leg if it exists in the global database.

Clearly, the differentiation between node and data variables in dtgds/degds can be seen as just syntactic sugar for standard relational schemas. The above restrictions (at most one node variable, at a fixed position, data-fullness) can then be seen as restrictions of classical constraints. In this sense, a dtgd like $R(x)@\kappa, S(x)@\mu \to T(x)@\kappa$ could be rewritten into a standard tgd of the form $R(\kappa, x), S(\mu, x) \to T(\kappa, x)$. The restriction to data-full dtgds thus translates to the restriction of existential quantification to these first attributes.

However, as the following example illustrates, our restriction to existential quantification of node variables does not translate into any of the restricted fragments with low complexity, which we are aware of.

▶ **Example 5.** Let Σ consist of a node-creating dtgd $R(x)@\kappa \to T(x)@\mu$ and a data-collecting dtgd $T(x)@\kappa, T(y)@\kappa, T(z)@\mu, T(w)@\mu \to U(x, y, z, w)@\kappa$. The corresponding set of standard tgds

$$\begin{aligned} R(\kappa, x) &\to T(\mu, x), \\ T(\kappa, x), T(\kappa, y), T(\mu, z), T(\mu, w) &\to U(\kappa, x, y, z, w), \end{aligned}$$

is neither sticky nor weakly guarded nor warded.[5] The set Σ has, however, bounded context (and its associated implication problem is shown to be in NP in Section 5).

Furthermore, the set consisting of $R(x,y)@\kappa \to S(x,y)@\mu$ and $S(x,x)@\kappa \to R(x,x)@\mu$ is not weakly acyclic but data-full with bounded context.

3.2 Examples of distribution constraints

In the following, we provide examples illustrating the versatility of distribution constraints. We begin with an examination of certain uses of distributed atoms. In principle, distributed atoms can be used in the body and in the head of constraints, referring to multiple node variables. Some more restricted uses seem particularly useful however.

We use the schema $\{\texttt{Emp}(\text{name}, \text{title}),\ \texttt{Sal}(\text{title}, \text{salary}),\ \texttt{Addr}(\text{name}, \text{address})\}$ as a running example for the remainder of this section.

3.2.1 Global dtgds and degds

We call distribution constraints *global* if they do not contain any distributed atom. These constraints refer to the global instance of a distributed database only – irrespective of the local databases. Formally, a dtgd (resp., degd) σ is a *global constraint* if σ is a tgd (resp., egd).

▶ **Example 6.** The following constraints are examples of a global dtgd and a global degd: $\texttt{Emp}(n,t) \to \texttt{Sal}(t,s)$, and $\texttt{Sal}(t,s), \texttt{Sal}(t,s') \to s = s'$. Together they specify that every employee has a unique salary.

3.2.2 Local dtgds and degds

Distribution constraints where every relation atom is a distributed atom and where all these atoms refer to the same node variable are called *local*. These constraints specify conditions that hold on *every* local instance, viewed on its own – irrespective of the global instance or other local instances.

▶ **Example 7.** The following is a local dtgd expressing that whenever a fact $\texttt{Emp}(a,b)$ occurs at node k there is a fact $\texttt{Sal}(b,c)$, for some element c in dom, that occurs at node k as well: $\texttt{Emp}(x,y)@\kappa \to \texttt{Sal}(y,z)@\kappa$. The following local value identifying degd expresses that, relative to each node, each employee (name) has a unique address. $\texttt{Addr}(x,y)@\kappa, \texttt{Addr}(x,y')@\kappa \to y = y'$.

3.2.3 Global-Local dtgds

Lastly, we call a dtgd *global-local* if none of its body atoms is distributed while all its heads atoms are distributed and refer to the same node variable.

▶ **Example 8.** The global-local constraint $\texttt{Emp}(x,y), \texttt{Sal}(y,z) \to \texttt{Emp}(x,y)@\kappa, \texttt{Sal}(y,z)@\kappa$ expresses that if there is an Emp-fact and a Sal-fact with the same title-attribute then these facts meet at some node. This means that the join condition between Emp and Sal induced by the schema is maintained in the horizontal decomposition of the global database.

[5] The set Σ' is not sticky because the marked variable μ occurs more than once in τ'. It is not (weakly) guarded because a single atom cannot contain both variables κ and μ that occur in affected positions of τ'. Finally, it is not warded because the dangerous variable κ appears in more than one atom in the body of τ'.

Global-local constraints can also express that the database has no skipped facts, i.e., facts $f \in \text{global}(D)$ with $f \notin \text{local}(D)$. To this end, for each relation symbol R a global-local constraint $R(x_1, \ldots, x_{\mathsf{ar}(R)}) \to R(x_1, \ldots, x_{\mathsf{ar}(R)})@\kappa$ can be added. Indeed, it is this ability of distributed constraints to disallow skipped facts that made us allow them in first place. For a schema $\mathcal{S}$, we denote by $\mathcal{U}(\mathcal{S})$ the set of all global-local constraints that express that there are no skipped facts.

By symmetry also local-global constraints can be defined. An example would be the dtgd $\texttt{Emp}(x, y)@\kappa, \texttt{Sal}(y, z)@\kappa \to \texttt{Addr}(x, z')$ (even though for this particular schema, the constraint is rather contrived). Nevertheless, local-global constraints allow to state explicitly that every local fact is also a global fact, by stating $R(x_1, \ldots, x_m)@\kappa \to R(x_1, \ldots, x_m)$, for every relation R (with $m = \mathsf{ar}(R)$).

3.3 Applications of distribution constraints

We give some applications of distribution constraints like defining range, hash and co-partitionings and testing for parallel-correctness. In the full version, we illustrate how hypercube distributions can be incorporated and discuss query answering and multi-round query evaluation. The paper [35] further explores the use of distribution constraints to model distributed evaluation strategies for Datalog in the context of parallel-correctness and parallel-boundedness in the multi-round MPC model.

3.3.1 Range and hash partitioning

Distribution constraints can easily incorporate the commonly used range and hash partitionings (see for example [31, 37]). Example 1 already illustrates range partitionings.

The following two distribution constraints define a hash partitioning of the relation $\texttt{Emp}(\text{name}, \text{dept})$ on the attribute department:

$$\begin{aligned} \texttt{Emp}(n, d) &\to \texttt{Emp}(n, d)@\kappa \\ \texttt{Emp}(n, d)@\kappa, \texttt{Emp}(n', d) &\to \texttt{Emp}(n', d)@\kappa \end{aligned}$$

The first rule enforces that every Emp-tuple occurs at a node while the second rule ensures that Emp-tuples within the same department are placed together. The above approach where hash functions are implicit should be contrasted with the modeling of Hypercube distributions, discussed in the full version, where hash functions are made explicit.

3.3.2 Co-partitioning

A popular way to avoid expensive remote join operations – already used in early parallel systems – is to co-partition tables on their join key [21, 25]. Generalizations of the latter technique where co-partitioning is determined by more complex join predicates have been shown to be effective in modern systems as well [38, 39, 41, 45].

Consider, for instance, the following (simplified) relations from the TPC-H schema [1]: $\texttt{Lineitem}(\text{linekey}, \text{orderkey})$, $\texttt{Orders}(\text{orderkey}, \text{custkey})$, and $\texttt{Customer}(\text{custkey}, \text{cname})$.

Zamanian, Binnig, and Salama [45] exemplify the following co-partitioning scheme: `Lineitem` is hash-partitioned by linekey, `Orders` tuples are co-partitioned with `Lineitem` tuples with the same orderkey, and `Customer` tuples are co-partitioned with `Orders` tuples with the same custkey. As a consequence, the join $\texttt{Lineitem} \bowtie \texttt{Orders} \bowtie \texttt{Customer}$ can be evaluated without expensive remote joins. We note that the work in [45] is by no means restricted to single-round or communication-free evaluation of queries. Knowledge of co-location of tuples is used to rewrite query plans and to determine those parts that can be evaluated without additional reshuffling. Partitionings are also not considered to be static but should adapt over time to changes in workload and the data (e.g., [32, 40]).

▶ **Example 9.** The following distribution constraints define the co-partitioning scheme mentioned above:

$$\texttt{Lineitem}(\ell, o) \rightarrow \texttt{Lineitem}(\ell, o)@\kappa \tag{1}$$

$$\texttt{Orders}(o, c) \rightarrow \texttt{Orders}(o, c)@\kappa \tag{2}$$

$$\texttt{Customer}(c, n) \rightarrow \texttt{Customer}(c, n)@\kappa \tag{3}$$

$$\texttt{Lineitem}(\ell, o), \texttt{Lineitem}(\ell, o')@\kappa \rightarrow \texttt{Lineitem}(\ell, o)@\kappa \tag{4}$$

$$\texttt{Lineitem}(\ell, o)@\kappa, \texttt{Orders}(o, c) \rightarrow \texttt{Orders}(o, c)@\kappa \tag{5}$$

$$\texttt{Orders}(o, c)@\kappa, \texttt{Customer}(c, n) \rightarrow \texttt{Customer}(c, n)@\kappa \tag{6}$$

Basically, Constraint (1) expresses that every `Lineitem` fact in the global database occurs at some node; similarly for Constraints (2) and (3). Constraint (4) then expresses that `Lineitem` facts are hashed on the first attribute: for every item ℓ with order o' stored on some server, every other order of that item is stored there too. Constraint (5) expresses that $\texttt{Orders}(o, c)$ facts are co-located with $\texttt{Lineitem}(\ell, o)$ facts, while Constraint (6) expresses that $\texttt{Customer}(c, n)$ facts are co-located with $\texttt{Orders}(o, c)$ facts. All together the distribution constraints imply that the join condition between the three relations is maintained in the horizontal decomposition of the global database.

3.3.3 Hierarchical partitioning schemes

Recent database systems like Google's F1 [39, 41] use hierarchical partitioning schemes to provide performance while ensuring consistency under updates. Hierarchical partitioning is a variant of the *co-partitioning* approach [42], introduced as *predicate-based reference partitioning* [45]. This approach allows to formulate a hashing condition for a relation S, given that a relation R is already distributed, in the following style: first, every S-fact has to be distributed, and second, if an S-fact joins with an R-fact on a predefined set of attributes, then the S-fact is distributed to every node where such an R-fact exists.

This is easily modeled by the following dtgds:

$$S(\boldsymbol{z}) \rightarrow S(\boldsymbol{z})@\kappa,$$

$$R(\boldsymbol{y})@\kappa, S(\boldsymbol{z}) \rightarrow S(\boldsymbol{z})@\kappa,$$

where variables $\boldsymbol{y}$ and $\boldsymbol{z}$ share some common variables $x_1, \ldots, x_n$ representing the join predicate. Notice that Example 9 follows this scheme. Another example, illustrating Google's AdWord scenario, is given in the full version.

3.3.4 Parallel-Correctness

We show that parallel-correctness of a conjunctive query can be captured by a dtgd but not always by a data-full one.

▶ **Example 10.** Let $Q = H(n, s) \leftarrow \texttt{Emp}(n, t), \texttt{Sal}(t, s)$ be a conjunctive query. Then, Q is parallel-correct on a distributed instance D if every fact from $Q(\text{global}(D))$ is derived at some node (due to the monotonicity of CQs, we do not need to check the converse statement). This can be expressed by the dtgd $\texttt{Emp}(x, y), \texttt{Sal}(y, z) \rightarrow \texttt{Emp}(x, y')@\kappa, \texttt{Sal}(y', z)@\kappa$. We note that in this dtgd κ and y' occur only in the head. So, this dtgd is *not* data-full.

4 Reasoning

We consider the implication problem for distribution constraints in Section 4.1, and adapt the chase to degds and data-full dtgds in Section 4.2, as a means to solve it.

4.1 The implication problem

We stress that the implied dependency τ in the definition below, is not required to belong to the class $\mathcal{C}$. That is, τ can be an arbitrary distribution constraint.

▶ **Definition 11.** *The* implication problem $\textsc{Imp}(\mathcal{C}, d)$*, parameterised by a class $\mathcal{C}$ of dependencies and a domain d asks, for a finite set Σ from $\mathcal{C}$ and a single distribution constraint τ, whether $\Sigma \models_d \tau$. Possible choices for d are $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$. If the choice of d does not matter or is clear from the context, we also write $\textsc{Imp}(\mathcal{C})$. For each $\alpha \geq 1$, we denote by $\textsc{Imp}_\alpha(\mathcal{C}, d)$ the restriction of $\textsc{Imp}(\mathcal{C}, d)$ to inputs (Σ, τ) in which the arity of each relation symbol (with respect to data) is at most α.*

Since every tgd is a dtgd and the implication problem for tgds without comparison atoms is undecidable, we instantly get the following.

▶ **Observation 12** ([13, 19])**.** *$\textsc{Imp}(\mathcal{T}_{all})$ is undecidable.*

To facilitate automatic reasoning, it thus makes sense to consider restricted kinds of constraints. An immediate observation is that most of the examples in Section 3 only use data-full distribution constraints. In the remainder of this paper, we therefore restrict our attention to this class.

▶ Remark 13. A full tgd $\sigma = \mathcal{A}, \mathcal{C} \rightarrow \{A'_1, \ldots, A'_p\}$ can be transformed into an equivalent set of tgds $\{\mathcal{A}, \mathcal{C} \rightarrow A'_1, \ldots, \mathcal{A}, \mathcal{C} \rightarrow A'_p\}$ with a singleton head.[6]In particular, this applies to dtgds *without existential quantification.* Similarly, data-full dtgds *with existentially quantified node variables* can be decomposed into data-full dtgds with at most one node variable in their head. We thus assume w.l.o.g. in *upper bound proofs* that all dtgds in Σ are decomposed in this fashion.

Since data-full dtgds have at most one node variable in their head, we can distinguish three kinds of data-full dtgds:

- *node-creating dtgds* like $R(x, y) \rightarrow R(x, y)@\kappa$, in which the one node variable in their head is existentially quantified;
- *data-collecting dtgds* like $S(x)@\kappa, T(x)@\lambda \rightarrow T(x)@\kappa$, which are dtgds that have one distributed head atom without existential quantification (they collect facts in a local node); and,
- *global dtgds* like $R(x, y) \rightarrow U(x)$ or $R(x, y)@\kappa, T(x)@\kappa \rightarrow S(y)$ that have one head atom without node variable (they contribute global facts).[7]

For brevity, we sometimes refer to *generating* and *collecting* dtgds.

We call the unique node variable that occurs in the head of a node-creating or data-collecting dtgd σ the *head variable* of σ.

[6] This observation is used also used the context of normalised schema mappings [24, 28].
[7] The term *global dtgd* thus represents a superset of global and local-global dtgds from Section 3.2.

4.2 The chase for distribution constraints

The classical way for deciding implication of tgds and egds builds on the *chase* procedure. In the following, we adapt the chase from [23] for distribution constraints from $\mathcal{T}_{\text{df}}$ and $\mathcal{E}_{\text{all}}$.

▶ **Definition 14** (chase step). *A dtgd σ is* applicable *to a distributed instance D with a valuation W if W satisfies* body_σ *on D and there exists no valuation W' for σ identical to W on* body_σ *such that $W'(\mathsf{head}_\sigma) \subseteq D$. Furthermore, if σ is node-creating then $W(\kappa)$ must be a node k not occurring in D, where κ is the head variable of σ. The* result $\mathsf{chase}(c, D)$ *of applying $c = (\sigma, W)$ on D is the distributed instance $D' = D \cup W(\mathsf{head}_\sigma)$ and we write $D \xrightarrow{c} D'$.*

For instance, if a distributed database D consists of facts $R(a)@1$ and $S(a,b)@2$, then dependency $\sigma = R(x)@\kappa, S(x,y)@\lambda \to R(x)@\mu, S(x,y)@\mu$ is applicable to D, as witnessed by the valuation W where $W(x,y) = (a,b)$ and $W(\kappa,\lambda,\mu) = (1,2,3)$. Application leads to $D' = D \cup \{R(a)@3, S(a,b)@3\}$.

If σ is node-creating, we say that the chase step *generates* the new node $W(\kappa)$ with an initial set $W(\mathsf{head}_\sigma)$ of facts. If σ is data-collecting, we say that the chase step *collects* the facts from $W(\mathsf{head}_\sigma)$ in node $W(\kappa)$. If σ is global, we say that the chase step *targets* the global database. The chase step *contributes* the set $W(\mathsf{head}_\sigma)$ of global facts.

▶ **Definition 15** (chase sequence). *Let Σ be a set of distribution constraints and D a distributed instance. A* chase sequence *for D with Σ is a sequence $\mathbf{D} = D_0, D_1, \ldots$ of distributed instances with $D_0 = D$ and $D_i \xrightarrow{(\sigma_i, W_i)} D_{i+1}$, for every $i \geq 0$ and some $\sigma_i \in \Sigma$ and valuation W_i. We write* $\mathsf{chase}(\mathbf{D})$ *for the final instance of $\mathbf{D}$, if $\mathbf{D}$ is finite.*

A chase sequence $\mathbf{D}$ fails, *if there is a degd $\sigma \in \Sigma$ with a head $t = t'$ and a valuation W, such that W satisfies* body_σ *on* $\mathsf{chase}(\mathbf{D})$ *and $W(t) \neq W(t')$. It is* successful, *if it is finite, does not fail and* $\mathsf{chase}(\mathbf{D})$ *has no applicable chase step.*

The following easy observation is crucial for our results.

▶ **Proposition 16.** *For each distributed database D and each set Σ of constraints from $\mathcal{T}_{df}$ and $\mathcal{E}_{all}$, there are no infinite chase sequences for D.*

For classical tgds and egds, to test $\Sigma \models \tau$, the chase is basically applied to a "canonical database" $V(\mathsf{rbody}_\tau)$, for some one-one valuation V. Due to comparison atoms, this does not suffice in our setting. Instead, we consider a set of canonical databases, which allows for all possible linear orders on the variables of body_τ. It depends on the general domain d which we allow to be one of $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$. More precisely, it is defined over a set $\mathsf{dom}(\Sigma, \tau, d)$ of data values that contains all constants of Σ and τ and, between each pair of successive constants all intermediate values from d or as many intermediate values as there are variables in body_τ. The set of canonical databases then consists of all databases of the form $V(\mathsf{rbody}_\tau)$ for valuations whose range is in $\mathsf{dom}(\Sigma, \tau, d)$.

Towards a formal definition, $c_1 < \cdots < c_\ell$ denote the constants in $\Sigma \cup \{\tau\}$ and let m be the number of data variables in body_τ. If $d = \mathbb{Q}$ then $\mathsf{dom}(\Sigma, \tau, d)$ consists of $c_1, \ldots, c_\ell$, all values $c_1 - m, \ldots, c_1 - 1$, all values $c_\ell + 1, \ldots, c_\ell + m$ and all values of the form $c_i + \frac{j}{m+1}(c_{i+1} - c_i)$, for $i \in \{1, \ldots, \ell - 1\}$ and $j \in \{1, \ldots, m\}$. If $d = \mathbb{Z}$, it consists of $c_1, \ldots, c_\ell$, all values $c_1 - m, \ldots, c_1 - 1$, all values $c_\ell + 1, \ldots, c_\ell + m$ and all values of the form $c_i + j$, for $i \in \{1, \ldots, \ell - 1\}$ and $j \in \{1, \ldots, m\}$ with $c_i + j < c_{i+1}$. If $d = \mathbb{N}$, it is defined as for $d = \mathbb{Z}$ with the additional constraint that elements of the form $c_1 - j$ must be non-negative.

By $\mathcal{D}(\Sigma, \tau, d)$ we denote the set of all distributed databases $V(\mathsf{rbody}_\tau)$, for which V maps data variables to values in $\mathsf{dom}(\Sigma, \tau, d)$ and node variables one-one to an initial segment of (a disjoint copy of) the natural numbers.

The following result shows that, to decide implication, it suffices to apply the chase to all databases in $\mathcal{D}(\Sigma, \tau, d)$.

▶ **Proposition 17.** *Let $\Sigma \cup \{\tau\}$ be a set distribution constraints from $\mathcal{T}_{df}$ and $\mathcal{E}_{all}$ and let d be one of $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$. Then the following statements are equivalent.*

(1) $\Sigma \models_d \tau$.

(2) *For every database $D = V(\mathsf{rbody}_\tau)$ in $\mathcal{D}(\Sigma, \tau, d)$ and every successful chase sequence $\mathbf{D}$ for D with Σ, there is an extension V' of V that satisfies head_τ on $\mathsf{chase}(\mathbf{D})$.*

(3) *For every database $D = V(\mathsf{rbody}_\tau)$ in $\mathcal{D}(\Sigma, \tau, d)$ there exists a chase sequence $\mathbf{D}$ for D with Σ, that fails or for which there is an extension V' of V that satisfies head_τ on $\mathsf{chase}(\mathbf{D})$.*

The straightforward proof is given in the full version.

5 Complexity

In this section, we study the complexity of the implication problem for data-full dtgds (and arbitrary degds). In general, this problem turns out to be in EXPTIME, in fact as EXPTIME-complete. We then study restrictions of dtgds and degds that lower the complexity of the implication problem. In fact, we identify fragments whose implication problems are Π_2^p-complete (NP-complete without comparison atoms) or PSPACE-complete. To wrap up the picture, we finally identify fragments that already yield EXPTIME-hardness.

For most practical cases, the relevant complexity is Π_2^p or even NP: the former is the case, e.g., if the database schema (or at least its arity) is fixed and if the number of atoms (or at least the number of variables) is bounded by some a-priori constant. The latter is the case if, additionally, there are no comparison atoms. In particular, the "natural" generalisations of our examples have at most Π_2^p (or NP) complexity.

The first result of this section states that $\textsc{Imp}(\mathcal{T}_{\text{df}})$ is EXPTIME-complete. The upper bound is very simple but shows that the problem is not harder than implication of full (non-distributed) tgds. On the other hand, in the distributed setting, EXPTIME-hardness already holds for fixed schemas with small arity, whereas this problem is easily seen to be in Π_2^p for (non-distributed) full dependencies.

▶ **Theorem 18.** *$\textsc{Imp}(\mathcal{T}_{df} \cup \mathcal{E}_{all})$ is EXPTIME-complete. The lower bound already holds for a fixed schema of arity 2.*

Proof sketch. The lower bound follows from Theorem 26, which is shown in Subsection 5.3 and offers a collection of types of distributed constraints that make the implication problem EXPTIME-hard.

The upper bound uses Proposition 17. Since $\mathsf{dom}(\Sigma, \tau, d)$ has polynomial size in $|\Sigma| + |\tau|$, the set $\mathcal{D}(\Sigma, \tau, d)$ contains only exponentially many databases, from which the chase needs to start. Furthermore, each constraint in Σ can be applied at most an exponential number of times, since there are only exponentially many different valuations, and each of them can fire at most once. Therefore, each chase sequence is of at most exponential length. ◀

Intuitively, EXPTIME-hardness for the class $\mathcal{T}_{\text{df}}$ of data-full dtgds (and even without degds) follows from the need to keep track of an exponential number of nodes, as can be seen from the proof of the lower bound of Theorem 26.

In the following two subsections, we turn to restricted classes with lower complexity[8] for the implication problem. The fragments that we study are not motivated from practical considerations (since there we already have "low" complexity). They were rather obtained by considering syntactic properties under which the chase behaves better than in general.

First of all, these fragments require a fixed bound on the arity of relations. Furthermore, they bound the amount of data that is associated with a single node in a dtgd, in various ways. To state this more precisely, we use the following notions.

▶ **Definition 19** (bounded context). *The* context $\mathsf{cont}_\kappa(\mathcal{A})$ *of a node variable* κ *in a set* $\mathcal{A}$ *of atoms is the set of (data) variables occurring in atoms referring to* κ*. The context* $\mathsf{cont}_\kappa(\sigma)$ *of* κ *in a dtgd* σ *is* $\mathsf{cont}_\kappa(\mathsf{rbody}_\sigma \cup \mathsf{head}_\sigma)$*. The context* $\mathsf{cont}_\kappa(\sigma)$ *of* κ *in a degd* σ *is* $\mathsf{cont}_\kappa(\mathsf{rbody}_\sigma)$*.*

A node variable κ *has* b-bounded context *in* σ *if* $|\mathsf{cont}_\kappa(\sigma)| \leq b$*. It has* b-bounded body context *if* $|\mathsf{cont}_\kappa(\mathsf{rbody}_\sigma)| \leq b$*.*

For instance, in the two following constraints,

- dtgd $\sigma_1 = R(x,y,z), S(y)@\kappa \to T(x)@\kappa$ and
- degd $\sigma_2 = S(x)@\kappa, S(y)@\kappa, R(x,y,z)@\lambda \to \kappa = \lambda$,

node variable κ has context $\{x,y\}$ and thus 2-bounded context. The body context of κ in σ_1 is even 1-bounded. Note that, in σ_2, node variable λ has 3-bounded body context and that, since there is no other node variable, the body context of this constraint is bounded by $3 = \max\{2,3\}$ in general.

We sometimes simply speak of *bounded context* if b is clear from the, well, context.

5.1 Classes with Π_2^p-reasoning

In this subsection, we consider two fragments which allow reasoning in Π_2^p in general, and in NP, if there are no comparison atoms.

The first fragment requires only one restriction (besides the usual arity restriction). The *bounded generation* fragment $\mathcal{T}_{\text{bg}}^b$ allows all global and data-collecting dtgds but only node-creating dtgds, in which the head variable has b-bounded context. We refer to the latter as node-creating dtgds of Type (G1), cf. Table 1.

▶ **Theorem 20.** *For fixed* $\alpha \geq 1$ *and* $b \geq 1$*, problem* $\textsc{Imp}_\alpha(\mathcal{T}_{bg}^b \cup \mathcal{E}_{all})$ *is*

1. Π_2^p*-complete in general and*
2. *NP-complete, if restricted to inputs without comparison atoms.*

Proof idea. The lower bounds follow by reductions from the containment problem for conjunctive queries (with or without comparisons) [20, 44]. For two queries $\mathcal{Q}$ and $\mathcal{Q}'$ of the respective classes, query $\mathcal{Q}$ is contained in $\mathcal{Q}'$ if and only if $\{\sigma\} \models \tau$, where $\sigma = \mathsf{body}_{\mathcal{Q}'} \to \mathsf{head}_{\mathcal{Q}'}$ and $\tau = \mathsf{body}_{\mathcal{Q}} \to \mathsf{head}_{\mathcal{Q}}$ are considered as global data-collecting dtgds (with or without comparisons).

The proofs of the upper bounds use Condition (3) from Proposition 17 and rely on the fact that, in each chase sequence, thanks to the (G1)-restriction only a polynomial number of nodes is generated and thanks to the arity restriction, each can carry only a polynomial number of facts. The Π_2^p upper bound can be almost directly inferred from the quantifier structure of Condition (3). The NP upper bound follows since, essentially, only one initial database needs to be considered. Below, we provide the details.

[8] This statement holds under the common assumption that Π_2^p and PSPACE are smaller than EXPTIME.

We begin by showing that all possible chase sequences have polynomial length. Let Σ be a set of dependencies in $\mathcal{T}^b_{\text{bg}} \cup \mathcal{E}_{\text{all}}$ and τ be a distribution constraint. We recall that the chase applies a node-creating chase step with a dtgd σ and a valuation W only if no node with the facts from $W(\mathsf{head}_\sigma)$ exists. However, for each $\sigma \in \Sigma$, the number of variables in head_σ occurring in atoms related to κ is at most b and thus the number of different valuations of head_σ (with the initial values derived from D_τ) is at most $|\mathsf{dom}(\Sigma, \tau, d)|^b$, and thus polynomial. Therefore, the number of chase steps using a σ of Type (G1) is polynomially bounded. In particular, the chase generates only a polynomial number of nodes. Since data-collecting dtgds have only one atom in their head and α is a bound on the arity of atoms, there can only be a polynomial number of chase steps using data-collecting dtgds, for each node. Similarly, there can only be a polynomial number of chase steps using global dtgds. Altogether there can be only a polynomial number of chase steps in each chase sequence. As mentioned before, the Π^p_2 upper bound follows from Condition (3) in Proposition 17. Universal quantification is over all databases in $\mathcal{D}(\Sigma, \tau, d)$, the chase sequence $\mathbf{D}$ is existentially quantified and that it fails or there exists an appropriate extension can be verified by further existential quantification.

If there are no comparison atoms in Σ and τ it suffices to start the chase from *one* canonical database of the form $V(\mathsf{body}_\tau)$, for some one-one valuation V that does not map any variables of body_τ to constants of body_τ. However, the chase needs to be defined in a slightly different fashion: if a degd with a head of the form $t = t'$ is applicable via a valuation W then in the result $W(t)$ and $W(t')$ are identified, unless they are different constants from body_τ. If the latter is the case, the chase fails. For this version of the chase, Proposition 17 holds as well.

The NP upper bound then follows, since only one initial database needs to be used and only one chase sequence of polynomial length needs to be guessed. ◀

The other class of dtgds considered in this subsection allows node-creating dtgds with head variables with unbounded context. The simple argument of the proof of Theorem 20 therefore does not work anymore. However, it turns out that there are simple (and still generous) restrictions that guarantee a Π^p_2 (NP) upper bound for the implication problem. To this end, we define the *bounded context fragment* $\mathcal{T}^b_{\text{bc}}$ of dtgds as follows (cf. Table 1).

▶ **Definition 21** (bounded context dtgds). *A node-creating dtgd σ is in $\mathcal{T}^b_{bc}$, if*
(G1) *its head variable has b-bounded context, or*
(G2) *all node variables in its body have b-bounded context.*
A data-collecting dtgd σ is in $\mathcal{T}^b_{bc}$, if
(C1) *its head variable has b-bounded body context, or*
(C2) *all other node variables have b-bounded context.*
For instance, all global dtgds and degds are in $\mathcal{T}^b_{bc}$.

The *bounded context fragment* $\mathcal{E}^b_{\text{bc}}$ of degds is defined similarly.

▶ **Definition 22** (bounded context degds). *A degd σ with only data variables in its head is in $\mathcal{E}^b_{bc}$. A degd $\sigma = \mathcal{A} \to \kappa = \mu$ is in $\mathcal{E}^b_{bc}$, if*
(E1) *κ and μ have b-bounded context, or*
(E2) *μ and all node variables that do not occur in the head have b-bounded context.*
The degds of $\mathcal{E}^b_{\text{bc}}$ are illustrated in Table 1. In degds of Type (E2), we call κ (but not μ) the *head variable.*

We can now state the second result of this subsection.

▶ **Theorem 23.** *For fixed $\alpha \geq 1$ and $b \geq 1$, problem* $\text{IMP}_\alpha(\mathcal{T}^b_{bc} \cup \mathcal{E}^b_{bc})$ *is*

1. Π^p_2*-complete in general and*
2. *NP-complete, if restricted to inputs without comparison atoms.*

Proof idea. For the upper bounds, we show that any chase sequence for $\mathcal{T}^b_{\text{bc}} \cup \mathcal{E}^b_{\text{bc}}$ can be *normalised* such that only a polynomial number of *witness nodes* are needed to trigger any chase steps. Since every node has only a polynomial number of facts, this implies that it suffices to consider chase sequences of polynomial length. The remaining arguments are then as for Theorem 20. The lower bounds follow by the same reduction as in Theorem 20. ◀

5.2 Classes with PSPACE-reasoning

In this subsection, we consider a fragment of distribution constraints that does not guarantee polynomial-length chase sequences but, intuitively, sequences of polynomial "width". Consequently, their implication problem turns out as PSPACE-complete.

The fragment $\mathcal{T}^b_{\text{wbc}}$ is defined as follows (cf. Table 1).

▶ **Definition 24** (weakly bounded distribution tgds). *Let $b \geq 1$. A dtgd σ is in the class $\mathcal{T}^b_{wbc}$ of* weakly bounded distribution tgds *if it is in $\mathcal{T}^b_{\text{bc}}$ or it obeys the following restriction:*

(G3) σ is node-creating and exactly one of its node variables does not *have b-bounded body context.*

▶ **Theorem 25.**

1. $\text{IMP}_\alpha(\mathcal{T}^b_{wbc} \cup \mathcal{E}^b_{bc})$ *is in PSPACE, for every $\alpha \geq 1$ and $b \geq 1$.*
2. $\text{IMP}_\alpha(\mathcal{T}^b_{wbc})$ *is PSPACE-hard for $\alpha \geq 1$ and $b \geq 0$. This lower bound even holds without comparison atoms.*

Proof idea. The lower bound (2) is shown similarly as PSPACE-hardness of the implication problem for inclusion dependencies over schemas of *unbounded* arity [4, 18]. In a nutshell, in this reduction each node carries *one tuple*, encoded with unary relations.

For the upper bound, unlike for $\mathcal{T}_{\text{bc}}$, we do not have a polynomial length bound for chase sequences for $\mathcal{T}_{\text{wbc}}$. In fact, it might be the case that a chase sequence generates an exponential number of nodes. However, we can still use a polynomially bounded set Z of witness nodes for the bounded node variables of (G3) dtgds and for all other constraints. They do not account for the unbounded node variables in (G3) constraints, but we show that those only need to occur in linear succession. The basic idea of the algorithm is to guess Z (and the facts on nodes from Z) and to verify in polynomial space, for each node in Z, that it is produced by a chase sequence. These verifying computations all assume the same set Z. We use a kind of timestamps to avoid cyclic reasoning. The details of this proof are given in the full version. ◀

5.3 Classes with EXPTIME-hard reasoning

In this subsection, we turn to combinations of constraints that yield an EXPTIME-hard implication problem. In particular, we complete the proof of Theorem 18. To this end, we consider the following additional types of constraints:

(G4) node-creating dtgds with two unbounded node variables;
(C3) data-collecting dtgds with two unbounded node variables;
(E3) degds with two unbounded node variables; and,
(E4) degds with three unbounded node variables.

▶ **Theorem 26.** *$\textsc{Imp}(\mathcal{T}_{df})$ is EXPTIME-hard. This statement holds already without comparison atoms and with only the following combinations of constraint types allowed:*

(a) *Node-creating dtgds of Type (G2) and data-collecting dtgds of Type (C3);*
(b) *Node-creating dtgds of Type (G2) and (G4);*
(c) *Node-creating dtgds of Type (G2) and degds of Type (E4);*
(d) *Node-creating dtgds of Types (G2) and (G3), and degds of Type (E3).*

In all cases, schemas with (at most) binary relations suffice.

The four EXPTIME-hard fragments are illustrated in Table 1. The reductions use an alternating Turing machine with *linearly* bounded space.

5.4 Parallel-correctness revisited

We lift parallel-correctness to the setting of distribution constraints. In particular, we say that a query Q is *parallel-correct w.r.t. a set of distribution constraints* Σ if Q is parallel-correct on every database that satisfies Σ.

As parallel-correctness of a conjunctive query can be expressed as a dtgd, the results of the present section lead to the following:

▶ **Corollary 27.** *For a CQ Q and a set of distribution constraints Σ, the complexity of deciding parallel-correctness of Q w.r.t. Σ is in EXPTIME. Furthermore, it is in Π_2^p (or NP, without comparsion atoms) and PSPACE if $\Sigma \subseteq \mathcal{T}_{bc}^b \cup \mathcal{E}_{bc}^b$ and $\Sigma \subseteq \mathcal{T}_{wbc}^b \cup \mathcal{E}_{bc}^b$, respectively, for a fixed b and a fixed bound α on the maximal arity of relation symbols.*

6 Conclusion

In this work, we introduced a novel declarative framework based on classical tgds and egds with comparison atoms to specify and reason about classes of data distributions. We illustrated our framework by various examples and performed an initial study of the complexity of the implication problem. As an application, we derived bounds (in Corollary 27) for the complexity of parallel-correctness of conjunctive queries.

Of course, there are many immediate general directions for extending the line of work started in this paper. For instance, one could study the implication problem for more expressive distribution constraints than data-full ones. There is a plethora of work on fragments of dependencies for improving the complexity of decision problems (e.g., [10, 14, 16, 36]). It could be investigated if any of these or others lead to a decidable implication problem. Another direction for future work is to study parallel-correctness w.r.t. distribution constraints for more expressive query languages than conjunctive queries. Some possibilities are unions of conjunctive queries [7], conjunctive queries with negation [26] or Datalog [29].

Example 9 and Section 3.3.3 illustrate how co-partitioning schemes can be translated into distribution constraints. It would be interesting to investigate the converse direction. That is, by design, distribution constraints specify in a declaratively way which properties a horizontal partitioning should satisfy. They do not provide a direct operational way to compute an actual partitioning. A natural question is to find an optimal partitioning satisfying a given set of distribution constraints.

Section 3 mentions a translation of distribution constraints to classical tgds and egds by increasing the arity of relations by one to take the node variables into account. It would be interesting to see whether the resulting fragment of dependencies is worthwhile to study it on its own in the classical setting.

Table 1 Illustration of restricted classes of dtgds and degds. Node variables that have to be bounded are shaded, others may be unbounded. The columns indicate the complexity of (some) combinations of fragments.

		Π_2^p	(NP)	PSPACE	EXPTIME			
(G1)	$\boxed{\lambda_1} \cdots \boxed{\lambda_r} \rightarrow \colorbox{lightgray}{κ}$	✓	✓	✓				
(G2)	$\colorbox{lightgray}{$\lambda_1$} \cdots \colorbox{lightgray}{λ_r} \rightarrow \boxed{\kappa}$		✓	✓	✓		✓	✓
(G3)	$\colorbox{lightgray}{$\lambda_1$} \cdots \colorbox{lightgray}{λ_r} \boxed{\mu} \rightarrow \boxed{\kappa}$			✓				✓
(G4)	$\boxed{\lambda} \boxed{\mu} \rightarrow \boxed{\kappa}$					✓		
Unrestricted data-collecting dtgds		✓						
(C1)	$\colorbox{lightgray}{$\kappa$} \boxed{\lambda_1} \cdots \boxed{\lambda_r} \rightarrow \boxed{\kappa}$		✓	✓				
(C2)	$\boxed{\kappa} \colorbox{lightgray}{λ_1} \cdots \colorbox{lightgray}{λ_r} \rightarrow \boxed{\kappa}$		✓	✓				
(C3)	$\boxed{\kappa} \boxed{\lambda} \rightarrow \boxed{\kappa}$				✓			
Unrestricted degds		✓						
(E1)	$\colorbox{lightgray}{$\kappa$} \colorbox{lightgray}{$\mu$} \boxed{\lambda_1} \cdots \boxed{\lambda_r} \rightarrow \kappa = \mu$		✓	✓				
(E2)	$\boxed{\kappa} \colorbox{lightgray}{μ} \colorbox{lightgray}{λ_1} \cdots \colorbox{lightgray}{λ_r} \rightarrow \kappa = \mu$		✓	✓				
(E3)	$\boxed{\kappa} \boxed{\lambda} \rightarrow \kappa = \lambda$							✓
(E4)	$\boxed{\kappa} \boxed{\lambda} \boxed{\mu} \rightarrow \kappa = \lambda$						✓	
	Theorem	20	23	25	26(a)	26(b)	26(c)	26(d)

The main technical challenge left open from this work is whether the EXPTIME-hardness result in Theorem 26(c) can be extended to rules of Type (E4) that contain two rather than three unbounded node variables.

References

1 TPC-H. URL: `http://www.tpc.org/tpch/`.

2 Serge Abiteboul, Émilien Antoine, and Julia Stoyanovich. The Webdamlog System Managing Distributed Knowledge on the Web. *CoRR*, abs/1304.4187, 2013. `arXiv:1304.4187`.

3 Serge Abiteboul, Meghyn Bienvenu, Alban Galland, and Émilien Antoine. A rule-based language for web data management. In *PODS*, pages 293–304, 2011.

4 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: `http://webdam.inria.fr/Alice/`.

5 Foto N. Afrati, Chen Li, and Vassia Pavlaki. Data exchange in the presence of arithmetic comparisons. In *EDBT*, pages 487–498, 2008.

6 Peter Alvaro, William R. Marczak, Neil Conway, Joseph M. Hellerstein, David Maier, and Russell Sears. Dedalus: Datalog in Time and Space. In *Datalog Reloaded*, pages 262–281, 2010.

7 Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-Correctness and Transferability for Conjunctive Queries. *J. ACM*, 64(5):36:1–36:38, 2017. `doi:10.1145/3106412`.

8 Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014. URL: `http://www.cambridge.org/9781107016163`.

9 Alessandro Artale, Roman Kontchakov, Alisa Kovtunova, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. Ontology-Mediated Query Answering over Temporal Data: A Survey (Invited Talk). In *TIME*, pages 1:1–1:37, 2017.

10 Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Walking the Complexity Lines for Generalized Guarded Existential Rules. In *IJCAI*, pages 712–717, 2011.

11 Marianne Baudinet, Jan Chomicki, and Pierre Wolper. Constraint-Generating Dependencies. *J. Comput. Syst. Sci.*, 59(1):94–115, 1999. doi:10.1006/jcss.1999.1632.

12 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication Steps for Parallel Query Processing. *J. ACM*, 64(6):40:1–40:58, 2017.

13 Catriel Beeri and Moshe Y. Vardi. The Implication Problem for Data Dependencies. In *ICALP*, pages 73–85, 1981.

14 Michael Benedikt. How Can Reasoners Simplify Database Querying (And Why Haven't They Done It Yet)? In *PODS*, pages 1–15, 2018.

15 Gerald Berger, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. The Space-Efficient Core of Vadalog. *CoRR*, abs/1809.05951, 2018. arXiv:1809.05951.

16 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013. doi:10.1613/jair.3873.

17 Andrea Calì, Georg Gottlob, and Andreas Pieris. Advanced Processing for Ontological Queries. *PVLDB*, 3(1):554–565, 2010. doi:10.14778/1920841.1920912.

18 Marco A. Casanova, Ronald Fagin, and Christos H. Papadimitriou. Inclusion Dependencies and Their Interaction with Functional Dependencies. *J. Comput. Syst. Sci.*, 28(1):29–59, 1984.

19 Ashok K. Chandra, Harry R. Lewis, and Johann A. Makowsky. Embedded Implicational Dependencies and their Inference Problem. In *STOC*, pages 342–354, 1981.

20 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

21 D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H. . Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, 1990.

22 Deming Dou and Stéphane Coulondre. A sound and complete chase procedure for constrained tuple-generating dependencies. *J. Intell. Inf. Syst.*, 40(1):63–84, 2013. doi:10.1007/s10844-012-0216-5.

23 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005. doi:10.1016/j.tcs.2004.10.033.

24 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. In Catriel Beeri and Alin Deutsch, editors, *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 14-16, 2004, Paris, France*, pages 83–94. ACM, 2004. doi:10.1145/1055558.1055572.

25 Shinya Fushimi, Masaru Kitsuregawa, and Hidehiko Tanaka. An Overview of The System Software of A Parallel Relational Database Machine GRACE. In *VLDB*, pages 209–219, 1986.

26 Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-Correctness and Containment for Conjunctive Queries with Union and Negation. In *ICDT*, 2016.

27 Gaetano Geck, Frank Neven, and Thomas Schwentick. Distribution constraints: The chase for distributed data, 2020. arXiv:2003.00965.

28 Georg Gottlob, Reinhard Pichler, and Vadim Savenkov. Normalization and optimization of schema mappings. *VLDB J.*, 20(2):277–302, 2011. doi:10.1007/s00778-011-0226-x.

29 Bas Ketsman, Aws Albarghouthi, and Paraschos Koutris. Distribution Policies for Datalog. In *ICDT*, pages 17:1–17:22, 2018.

30 Bas Ketsman and Dan Suciu. A Worst-Case Optimal Multi-Round Algorithm for Parallel Computation of Conjunctive Queries. In *PODS*, pages 417–428, 2017.

31 Martin Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* O'Reilly, 2016. URL: `http://shop.oreilly.com/product/0636920032175.do`.

32 Yi Lu, Anil Shanbhag, Alekh Jindal, and Samuel Madden. AdaptDB: Adaptive Partitioning for Distributed Joins. *PVLDB*, 10(5):589–600, 2017.

33 Michael J. Maher and Divesh Srivastava. Chasing Constrained Tuple-Generating Dependencies. In *PODS*, pages 128–138, 1996.

34 Vera Zaychik Moffitt, Julia Stoyanovich, Serge Abiteboul, and Gerome Miklau. Collaborative Access Control in WebdamLog. In *SIGMOD*, pages 197–211, 2015.

35 Frank Neven, Thomas Schwentick, Christopher Spinrath, and Brecht Vandevoort. Parallel-Correctness and Parallel-Boundedness for Datalog Programs. In *ICDT*, pages 14:1–14:19, 2019.

36 Adrian Constantin Onet. The chase procedure and its applications. PhD Thesis, June 2012. URL: `https://spectrum.library.concordia.ca/974476/`.

37 M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, Third Edition.* Springer, 2011.

38 Wolf Rödiger, Tobias Mühlbauer, Philipp Unterbrunner, Angelika Reiser, Alfons Kemper, and Thomas Neumann. Locality-sensitive operators for parallel main-memory database clusters. In *ICDE*, pages 592–603, 2014.

39 Bart Samwel, John Cieslewicz, Ben Handy, Jason Govig, Petros Venetis, Chanjun Yang, Keith Peters, Jeff Shute, Daniel Tenedorio, Himani Apte, Felix Weigel, David Wilhite, Jiacheng Yang, Jun Xu, Jiexing Li, Zhan Yuan, Craig Chasseur, Qiang Zeng, Ian Rae, Anurag Biyani, Andrew Harn, Yang Xia, Andrey Gubichev, Amr El-Helw, Orri Erling, Zhepeng Yan, Mohan Yang, Yiqun Wei, Thanh Do, Colin Zheng, Goetz Graefe, Somayeh Sardashti, Ahmed M. Aly, Divy Agrawal, Ashish Gupta, and Shivakumar Venkataraman. F1 query: Declarative querying at scale. *PVLDB*, 11(12):1835–1848, 2018. URL: `http://www.vldb.org/pvldb/vol11/p1835-samwel.pdf`, `doi:10.14778/3229863.3229871`.

40 Marco Serafini, Rebecca Taft, Aaron J. Elmore, Andrew Pavlo, Ashraf Aboulnaga, and Michael Stonebraker. Clay: Fine-Grained Adaptive Partitioning for General Database Schemas. *PVLDB*, 10(4):445–456, 2016.

41 Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, John Cieslewicz, Ian Rae, Traian Stancescu, and Himani Apte. F1: A distributed SQL database that scales. *PVLDB*, 6(11):1068–1079, 2013. `doi:10.14778/2536222.2536232`.

42 Bruhathi Sundarmurthy, Paraschos Koutris, and Jeffrey F. Naughton. Exploiting Data Partitioning To Provide Approximate Results. In *BeyondMR@SIGMOD*, pages 5:1–5:5, 2018.

43 Balder ten Cate, Phokion G. Kolaitis, and Walied Othman. Data exchange with arithmetic operations. In *EDBT*, pages 537–548, 2013.

44 Ron van der Meyden. The Complexity of Querying Indefinite Data about Linearly Ordered Domains. *J. Comput. Syst. Sci.*, 54(1):113–135, 1997. `doi:10.1006/jcss.1997.1455`.

45 Erfan Zamanian, Carsten Binnig, and Abdallah Salama. Locality-aware Partitioning in Parallel Database Systems. In *SIGMOD*, pages 17–30, 2015.