Theory of Computing Systems (2020) 64:965–998
https://doi.org/10.1007/s00224-019-09959-3

Distribution Policies for Datalog

Bas Ketsman[1] · **Aws Albarghouthi**[2] · **Paraschos Koutris**[2]

Published online: 4 December 2019

Abstract
Modern data management systems extensively use parallelism to speed up query processing over massive volumes of data. This trend has inspired a rich line of research on how to formally reason about the parallel complexity of join computation. In this paper, we go beyond joins and study the parallel evaluation of recursive queries. We introduce a novel framework to reason about multi-round evaluation of Datalog programs, which combines implicit *predicate restriction* with *distribution policies* to allow expressing a combination of data-parallel and query-parallel evaluation strategies. Using our framework, we reason about key properties of distributed Datalog evaluation, including parallel-correctness of the evaluation strategy, disjointness of the computation effort, and bounds on the number of communication rounds.

Keywords Datalog queries · Distributed evaluation · Distribution policies

1 Introduction

Modern data management systems—such as Spark [32, 38], Hadoop [13, 18], and others [19]—have extensively used parallelism to speed up query processing over massive volumes of data. Parallelism enables the distribution of computation into

This article belongs to the Topical Collection: *Special Issue on Database Theory (2018)*
Guest Editor: Benny Kimelfeld

✉ Bas Ketsman
bas.ketsman@vub.be

Aws Albarghouthi
aws@cs.wisc.edu

Paraschos Koutris
paris@cs.wisc.edu

[1] Vrije Universiteit Brussel, Brussels, Belgium

[2] University of Wisconsin-Madison, Madison, WI, USA

multiple servers, and thus significantly reduces the completion time for several critical data processing tasks. This trend has inspired a rich line of research on how to formally reason about the parallel complexity of join computation, one of the core tasks in massively parallel systems. Several papers [9, 10, 22–24] have studied the trade-off between synchronization (number of rounds) and communication cost, and have proposed and analyzed known and new parallel algorithms [4, 11]. Among these, the *Hypercube algorithm* [4, 16] can compute any multiway join query in one round by properly distributing the input data.

To reason about Hypercube-like algorithms, Ameloot et al. [6, 7] recently introduced a framework that captures one-round evaluation of joins under different data distributions. Their framework implicitly describes a single-round parallel algorithm through a *distribution policy*, which specifies how the facts in the input relations are distributed among the servers. While for non-recursive queries a distribution policy defines a scalable parallel evaluation strategy, for Datalog programs this is typically not the case. For instance, a simple transitive closure query already requires that for each component of the input database there exists a server containing all facts of the component.

To reason about Datalog evaluation in a distributed setting, we introduce a general theoretical framework that allows a combination of data and query parallelization strategies. The central concept in this framework is the notion of an *economic policy*. Our key observation is that, in order to deal with intensional predicates, we need to specify not only where a fact must be located to be *consumed* by a rule, but also where a fact must be *produced* by evaluating a rule of the program. An economic policy in our framework is defined as a pair of distribution policies: a *consumption policy*, which specifies the location of the facts that are used in the body of rules, and a *production policy*, which specifies the location of facts that appear in the head of a rule. The evaluation strategy that is implicitly defined by the data distribution must communicate any produced facts to the servers where they will be consumed, and thus can run over multiple rounds.

Our framework is inspired by a rich line of research on parallel evaluation strategies for Datalog programs from the early 90's [16, 35, 36, 39]. There, Datalog evaluation strategies are based on the idea of partitioning the instantiations of the program rules among servers by adding conditions to the bodies of the rules, called program restrictions. Some of the strategies proposed require no communication of intermediate (intensional) facts and thus can be completed in one round; other strategies require communication over multiple rounds. We show that an economic policy can capture several algorithms used for parallel evaluation of recursive and non-recursive queries, including the Hypercube algorithm [4, 16], and the decomposable strategies based on program restrictions [35].

In this framework we study several properties of economic policies. We first explore the property of *parallel-correctness*: when does an economic policy lead to a correct evaluation strategy? As can be expected, it is undecidable to show parallel-correctness for a general Datalog program, even for the simplest of economic policies. We therefore identify a sufficient condition: every minimal valuation of a rule must be *supported* by the policy. A rule valuation is supported if some server consumes all the facts in the body, and produces the fact in the head. For unions of

conjunctive queries, this condition is also necessary, recovering the result of Ameloot et al. [7]; however, we show that even for non-recursive programs with intermediate relations, the condition is no longer required. To overcome the undecidability of parallel-correctness, we identify a general family of economic policies, called *Generalized Hypercube Policies (GHPs)*, which are always parallel-correct, and further capture several commonly used parallel evaluation strategies.

Second, we study the property of *boundedness*: can we decide whether a given economic policy terminates in k rounds, independent of the input size? We show that there exists a sharp increase in complexity as we move from $k = 1$ to $k \geq 2$. For $k = 1$, we can succinctly characterize the structure of a policy that always terminates in one step. Additionally, given a GHP, we can do this in polynomial time in the description of the GHP. On the other hand, for $k \geq 2$ it is undecidable to determine whether it terminates in at most k steps, even for a GHP. We then ask which Datalog programs admit economic policies that are bounded by one round: we show that such programs are characterized by a syntactic property called *pivoting*, which was also identified by Wolfson and Silberschatz [37] in the context of decomposable programs.

The present paper is the full version of the extended abstract [21] and provides the missing proofs.

2 Related Work

2.1 Parallel Complexity

The parallel complexity of Datalog was first investigated by Cosmadakis and Kanellakis [12, 20]. Later work used the complexity class NC to theoretically capture which Datalog programs are efficiently parallelizable. Since Datalog evaluation is P-complete and the question whether P equals NC is a longstanding open problem, it is not known if every Datalog program belongs in NC, which implies that certain Datalog programs may not be significantly sped up through parallelism. Ullman and Van Gelder [33] showed that if a Datalog program has the *polynomial fringe property*, which says that every fact in the output has a proof tree of polynomial size, evaluation is in NC. Every linear Datalog program has the polynomial fringe property and is thus in NC. Afrati and Papadimitriou [3] showed that for simple chain queries (including non-linear queries) evaluation is either in NC or P-complete. Recently, Afrati and Ullman [5] studied the trade-off between communication and number of rounds. They describe a very restricted class of Datalog programs where it is possible to reduce the number of recursion steps (to a number that is logarithmic in the size of the input) without significantly increasing the communication cost.

2.2 Decomposability

The concept of predicate decomposability was first introduced by Wolfson and Silberschatz [37]. A predicate T is decomposable if there are $r > 1$ restricted copies $P_1, P_2, \ldots, P_r$ of the Datalog program P (using arithmetic predicates) such that

(i) the copies compute a partition of T for every input, and (ii) there exists an input instance where each copy will produce tuples over T. The main result is that decomposability is equivalent to pivoting for sirups where there are no constants, no repeating variables, and the sirup is linear or a simple chain rule. Here, a *sirup* is a Datalog program with one intensional predicate S and two rules: (i) a base rule $S(\boldsymbol{x}) \leftarrow B(\boldsymbol{x})$, and (ii) a recursive rule with head predicate S. A sirup is *linear* if S appears exactly once in the body of the recursive rule.

Later works [35, 36] redefine the concept of decomposability semantically. A Datalog program is *decomposable* if it is possible to partition the output domain (to at least two blocks) such that for every instance I, every output fact has a proof tree where all the intensional database facts belong in the same partition block. Wolfson and Ozen [36] show that deciding whether a given Datalog program is decomposable is undecidable. Cohen and Wolfson [35] provide necessary and sufficient syntactic conditions for decomposability for sirups where the arity of the intensional predicate is ≤ 2. They also define the notion of *strongly decomposable* sirups, where the partition must guarantee that, for some input, at least two blocks will produce a fact using the recursive rule of the sirup. Following the same line of work, Zhang et al. [39] present a more general framework that constructs partitionings of the rule instantiations. A related notion has also been studied by Ameloot et al. [8] in the context of connected Datalog programs.

2.3 Other Parallel Schemes

In addition to decomposability, several frameworks for parallel recursive processing were introduced in the early 90s [16, 35, 36]. Wolfson [35] generalizes decomposability to *load sharing schemes*, by allowing the output of a predicate to have overlap in the copies of the program P. Under a load sharing scheme, every linear program can be parallelized, even if it is not pivoting. In [15, 16, 36], general schemes are introduced that parallelize the evaluation by partitioning the set of rule instantiations, and allowing for communication among the servers (decomposable and load sharing schemes need no communication). Dewan et al. [14] proposes similar techniques with dynamic adjustments, to balance the load of a computation. Our framework differs in that the set of rule instantiations is distributed implicitly among the servers, according to the production and consumption policies, and that the communication between servers is made explicit.

2.4 Systems

Recent work studies the implementation of Datalog (or fragments of Datalog) on modern shared-nothing distributed systems. Seo et al. [29] present a distributed version of a Datalog variant for social network analysis called Socialite; however, their framework requires that the user provides annotations to guide the distribution of data. Wang et al. [34] implement a variant of Datalog on the Myria system [19], focusing mostly on asynchronous evaluation and fault-tolerance. The BigDatalog system [31] describes an implementation of Datalog on Apache Spark, but focuses mostly on linear Datalog programs that use aggregation. The task of parallelizing

Datalog has also been studied in the context of the popular MapReduce framework [2, 5, 30]. Motik et al. [26] provide an implementation of parallel Datalog in main-memory multicore systems.

3 Preliminaries

We assume an infinite domain **dom**. A database schema σ is a finite set of relation names $\{R_i\}_{i=1}^{n}$ with associated arities $ar(R_i)$. We shall write $R^{(k)}$ to denote a relation R with arity k. A fact $R(a_1, \ldots, a_k)$ is a tuple consisting of a relation name and a sequence of values from **dom**. We say that $R(a_1, \ldots, a_k)$ is *over* schema σ, if $R^{(k)} \in \sigma$. For a schema σ, we denote by $\mathsf{facts}(\sigma)$ the complete set of facts over σ. An *instance* I over σ is defined as a finite subset of $\mathsf{facts}(\sigma)$. We write $I_{|\sigma}$ to denote the subset of I containing all facts in I that are over schema σ.

For $i \in \mathbb{N}$, we abbreviate the set $\{1, \ldots, i\}$ by $[i]$, and for a set S we denote by $\mathcal{P}(S)$ its powerset.

3.1 Datalog

We assume an infinite domain of variables **var**, disjoint from **dom**. An *atom* is a formula $R(t_1, \ldots, t_k)$ consisting of a relation name and a tuple of terms; a *term* t_i is either a variable from **var** or a constant from **dom**.

A *Datalog rule* τ is of the form $R(\boldsymbol{x}) \leftarrow S_1(\boldsymbol{y_1}), \ldots, S_n(\boldsymbol{y_n})$, where $R(\boldsymbol{x})$ is a single atom called the head of τ, denoted $head_\tau$, and all $S_i(\boldsymbol{y_i})$ are atoms called body atoms of τ, denoted $body_\tau$. We say that $S_i(\boldsymbol{y_i})$ is *over* schema σ, when $S_i \in \sigma$ and $\boldsymbol{y_i}$ is a tuple of $ar(S_i)$ terms. We say that τ is over schema σ if all its atoms are. We assume that Datalog rules are always *safe*, i.e., that all variables in the head occur in at least one body atom. By $vars(\tau)$ we denote the set of variables in rule τ.

A *Datalog program* P is a finite set of Datalog rules. A program P is said to be over schema σ if all its rules are. Particularly, by $\textsc{edb}(P) \subseteq \sigma$ we denote the relation names occurring only in the body of rules, and by $\textsc{idb}(P) \subseteq \sigma$ all other relation names occurring in P. We further distinguish the names in $\textsc{idb}(P)$ by calling some of them *output relations*, denoted $out(P) \subseteq \textsc{idb}(P)$; all other intensional relations are *auxiliary*. We write $\sigma(P)$ to denote $\textsc{edb}(P) \cup \textsc{idb}(P)$.

Consider the directed graph whose nodes are the intensional relation names, and there is an edge from S to S' if S' occurs in the head of some rule τ of P, and S in the body of τ. We say that P is *recursive* if the graph is cyclic; otherwise, we say it is *non-recursive*. A non-recursive Datalog program with only one rule is called a *conjunctive query* (CQ).

3.2 Evaluation Semantics

We define the evaluation semantics of Datalog programs as usual, through the immediate consequence operator. Let P be a Datalog program and I an instance over $\textsc{edb}(P)$. A *valuation* v for rule $\tau \in P$ is a constant-preserving mapping of the terms in τ to values in **dom**. For a rule $\tau \in P$ and valuation v, we say that τ derives fact

$v(head_\tau)$ over instance I if $v(body_\tau) \subseteq I$. We refer to $v(\tau)$ as the instantiation of rule τ with valuation v.

We use T_P to denote the *immediate consequence operator* for P, which applies all rules in P exactly once over a given instance and adds all derived facts to that instance. Formally,

$$T_P(I) = I \cup \{v(head_\tau) \mid \tau \in P, \text{ valuation } v \text{ s.t. } v(body_\tau) \subseteq I\}.$$

Then, $P(I)$ is defined as the fixpoint reached after iteratively applying the immediate consequence operator over I. It is not difficult to see that T_P is monotone, and thus always reaches a fixpoint after a finite number of iterations. Moreover, the output of the query that P computes is defined as $P(I)_{|out(P)}$. We refer to Abiteboul et al. [1] for a detailed description.

We call a fact $\boldsymbol{f}$ *P-derivable* if $\boldsymbol{f} \in P(I)$ for some instance I, and *P-consumable* if during the evaluation of P on some instance I a rule instantiation $v(\tau)$ fires that requires $\boldsymbol{f}$. Both notions naturally generalize to atoms and relation names, e.g., relation name R is said to be P-consumable if some P-consumable fact $\boldsymbol{f}$ exists with symbol R. Atom A is P-consumable if a rule instantiation as above exists, with $A \in body_\tau$.

3.3 Proof Theoretic Concepts

Let $T = (V, E)$ be a tree. By $fringe_T$ we denote its leaves and by $root_T$ its root vertex. All other vertices are called internal vertices. For a vertex $n \in V$ we denote by $children_t(n)$ the set of child vertices of n in T. We now recall the classical notion of proof tree [1]. A *proof tree* T for a fact $\boldsymbol{f}$ on instance I and Datalog program P is a tree T with vertices over $\mathsf{facts}(\sigma(P))$, where $fringe_T \subseteq I$, $root_T = \boldsymbol{f}$, and for every internal vertex $\boldsymbol{g}$, there is a rule $\tau \in P$ and valuation v such that $\boldsymbol{g} = v(head_\tau)$ and $children_t(g) = v(body_\tau)$. In this case, we shall say that T *uses* the instantiation of τ with valuation v. It is easy to see that $P(I)$ consists of exactly those facts $\boldsymbol{f}$ for which a proof tree for $\boldsymbol{f}$ on I and P exists. We say that a rule instantiation $v(\tau)$ is *useless* if $v(head_\tau) \in v(body_\tau)$; otherwise, we say that it is *useful*. W.l.o.g. we will consider only proof trees where all rule instantiations are useful.

We say that a proof tree T' is *subsumed* by proof tree T for P, denoted $T' \sqsubseteq T$, if $fringe_{T'} \subseteq fringe_T$ and $root_{T'} = root_T$.[1]

4 The Framework

Our framework considers a setting with p *servers* that share no memory and can communicate only via messages—this is commonly referred to as a *shared-nothing* parallel architecture. The set of servers forms a *network* $[p]$ that we assume is fully connected. In order to define how computation is performed, we will use policies that

[1] Notice that in [21] we used the term *entailed*.

specify how the data (input and output facts) are distributed over the network. We borrow the definition of a distribution policy from [7]:

Definition 1 (**Distribution Policy**) A *distribution policy* $\boldsymbol{P} = (\mathsf{facts}_{\boldsymbol{P}})$ over schema σ and network $[p]$ consists of a function $\mathsf{facts}_{\boldsymbol{P}} : [p] \to \mathcal{P}(\mathsf{facts}(\sigma))$ mapping servers to sets of facts over σ.

Distribution policies are instance independent, i.e., they are oblivious of the specific database instance. Intuitively, a policy expresses on which servers a fact should reside if the fact is in the network, but not whether the fact is in the network. Henceforth, we slightly abuse notation and write $\boldsymbol{P}(\boldsymbol{f})$ to denote the set of servers *responsible* for $\boldsymbol{f}$, i.e., $\boldsymbol{P}(\boldsymbol{f}) = \{i \mid \boldsymbol{f} \in \mathsf{facts}_{\boldsymbol{P}}(i)\}$.

In contrast to [7], where the focus is on single-round query evaluation and policies that define only the initial data distribution over extensional database facts, we consider a multi-round setting that allows the communication of intermediate facts.

Definition 2 (**Economic Policy**) An *economic policy* $\boldsymbol{E}$ over schema σ and network $[p]$ is a pair $(\boldsymbol{P}, \boldsymbol{C})$ of distribution policies over the same universe U, where:

- $\boldsymbol{P}$ is defined over $\text{IDB}(P)$ and is called the *production policy*; and
- $\boldsymbol{C}$ is defined over $\text{EDB}(P) \cup \text{IDB}(P)$ and is called the *consumption policy*.

A production policy describes which servers have the responsibility of producing a certain intensional database fact. A consumption policy describes which servers need an extensional or intensional database fact to satisfy the body of a rule instantiation. We say that a fact $\boldsymbol{f}$ is $\boldsymbol{C}$-*consumable* if $\boldsymbol{C}(\boldsymbol{f}) \neq \emptyset$ and that relation R is $\boldsymbol{C}$-*consumable* if some fact over **dom** and R is $\boldsymbol{C}$-consumable.

A *family* of economic policies $\mathcal{F}$ is a set of economic policies over a common universe and schema. We say that a family $\mathcal{F}$ satisfies property $\mathcal{P}$ if all the policies in $\mathcal{F}$ satisfy $\mathcal{P}$.

4.1 Datalog Evaluation Modulo Policies

Instead of letting a server compute the full program over its local instance, we *restrict* the evaluation process based on a server's economic policy. That is, for economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ and Datalog program P, the following sequential evaluation algorithm takes place on server i:

- First, every rule $\tau \in P$ is annotated with policy-predicates as follows. For the head $R(\boldsymbol{x})$, we add a predicate $\mathit{Policy}_R^{\boldsymbol{P}}(\boldsymbol{x})$ to the body of τ. Here, relation name $\mathit{Policy}_R^{\boldsymbol{P}}$ refers to relation $\mathsf{facts}_{\boldsymbol{P}}(i)_{|\{R\}}$.
- Second, for every atom $S(\boldsymbol{y})$ in the body of τ, we add the predicate $\mathit{Policy}_S^{\boldsymbol{C}}(\boldsymbol{y})$, where now $\mathit{Policy}_S^{\boldsymbol{C}}$ refers to the relation $\mathsf{facts}_{\boldsymbol{P}}(i)_{|\{S\}}$.

The added predicates may be infinitely large, but can be accessed through queries of the form "$\boldsymbol{t} \in \mathsf{facts}_{\boldsymbol{P}}(i)_{|\{R\}}$?" or "$\boldsymbol{t} \in \mathsf{facts}_{\boldsymbol{C}}(i)_{|\{S\}}$?".

Throughout the paper, we assume the semi-naive evaluation strategy for Datalog programs. Semi-naive evaluation proceeds as usual over the annotated program: after each application of the fixpoint operator, the newly derived facts are added to a delta relation, and a rule instantiation is triggered only if at least one of its facts is in the delta relation from the previous iteration. We denote by $P_{\uparrow \boldsymbol{E}}(I, J)$ the fixpoint instance when we execute P restricted to $\boldsymbol{E}$ on input I, with delta relations initialized with J.

4.2 Distributed Evaluation Strategy

We now present how an economic policy induces a parallel evaluation strategy. Our parallel model is the BSP-based Massively Parallel Communication Model (MPC) [25]. In this model, computation is performed over servers in a multi-round fashion. Each round has two distinct phases: a local computation phase, and a synchronized communication phase.

Consider a Datalog program P, a network $[p]$, and an economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$. Moreover, let I be the input instance, which we initially assume to be partitioned arbitrarily over the p servers. Denote by local_i^0 the initial local instance of server i. Let local_i^k be the instance on server i right after the k-th communication phase.

At the k-th round (for $k \geq 1$), we perform the following procedure:

1. *Communication:* Every server sends its facts as defined by the consumption policy $\boldsymbol{C}$. That is, server i sends local fact $\boldsymbol{f} \in \mathsf{local}_i^{k-1}$ to server j if (and only if) $\boldsymbol{f} \in \mathsf{fact}_{\boldsymbol{C}}(j)$. Let rec_i^k be the facts received by server i during the k-th communication phase.[2]
2. *Computation:* Every server computes the local fixpoint: if $k = 1$, then $\mathsf{local}_i^k = P_{\uparrow \boldsymbol{E}}(\mathsf{rec}_i^k \cup \mathsf{local}_i^{k-1}, \mathsf{rec}_i^k \cup \mathsf{local}_i^{k-1})$, otherwise $\mathsf{local}_i^k = P_{\uparrow \boldsymbol{E}}(\mathsf{rec}_i^k \cup \mathsf{local}_i^{k-1}, \mathsf{rec}_i^k \setminus \mathsf{local}_i^{k-1})$.

Intuitively, the algorithm terminates when, after a round is finished, for every server all locally derived facts that need to be sent to some other server according to the consumption policy, were already sent to these servers in an earlier round.

Formally, for server i, we define set $F_i = \{\boldsymbol{f} \mid \boldsymbol{C}(\boldsymbol{f}) \setminus \{i\} \neq \emptyset\}$. Intuitively, F_i represents all facts consumed by servers other than i itself. We say that a server has reached a *local fixpoint state* for $\boldsymbol{E}$ and P after round $k \geq 1$, if $\mathsf{local}_i^k \cap F_i \subseteq \mathsf{local}_i^{k-1}$. We say that the network $[p]$ has reached a *global fixpoint state* for $\boldsymbol{E}$ and P after round k, if all servers $i \in [p]$ have reached a local fixpoint state after round k. Notice that this condition is as desired, because every round goes into the communication phase first, then into the local computation phase. Hence, all earlier sent messages have been taken into account.

One could imagine a smarter communication procedure that incorporates Datalog semantics as well. For example, a server does not need to send a local fact $\boldsymbol{f} \in$

[2] We remark that from a practical viewpoint it makes no sense to communicate the same facts more than once. When $j = i$, no actual communication takes place.

$\mathsf{facts}_{\boldsymbol{C}}(j)$ to server j if for every input I server j is guaranteed to already have $\boldsymbol{f}$ in its local instance. However, it is in general undecidable to make such a decision (see Lemma 2).

For an instance I, let $[P, \boldsymbol{E}](I)$ denote the union of all facts over $out(P)$ found at any server after reaching the global fixpoint. Notice that the above evaluation strategy always reaches a fixpoint, due to monotonicity of Datalog.

Example 1 Consider the left-linear Datalog program that computes transitive closure:

$$T(x, y) \leftarrow R(x, y). \qquad T(x, y) \leftarrow T(x, z), R(z, y).$$

For any function $h : \mathbf{dom} \to [p]$, we define the economic policy $(\boldsymbol{P}_1, \boldsymbol{C}_1)$, where $\boldsymbol{C}_1(R(a, b)) = [p]$, and $\boldsymbol{C}_1(T(a, b)) = \boldsymbol{P}_1(T(a, b)) = \{h(a)\}$ for all $a, b \in \mathbf{dom}$. This policy works as follows: it replicates the extensional database facts everywhere, and then produces/consumes each fact $T(a, b)$ at server $h(a)$. It is easy to see that the economic policy correctly computes the transitive closure. In fact, the evaluation always terminates in a single round.

Consider a different policy $(\boldsymbol{P}_2, \boldsymbol{C}_2)$, which again takes any function $h : \mathbf{dom} \to [p]$ and which has the following definition:

$$\begin{aligned} \boldsymbol{C}_2(R(a, b)) &= \{h(a)\}, \\ \boldsymbol{C}_2(T(a, b)) &= \{h(b)\}, \text{ and} \\ \boldsymbol{P}_2(T(a, b)) &= [p]. \end{aligned}$$

This policy does not replicate the extensional database facts, but it hash-partitions them according to the first attribute. Whenever a server discovers a new fact, the new fact has to be consumed to the location determined by the hash of the second attribute. Observe that the production policy is $[p]$ because we do not know where each fact will be produced (in other words, each server will produce as many intensional database facts as possible from its local input without any restrictions).

We will see later in Section 6 that all the above economic policies belong in a specific family of policies that we call Generalized Hypercube Policies (GHPs). We notice that our framework supports evaluation strategies that are *oblivious* of the instance: each fact is communicated, consumed, and produced independent of whether other facts are in the same local instance or not. Lastly, we note that monotonicity of Datalog ensures monotonic behaviour of economic policies for Datalog programs, as made formal by Proposition 1.

Proposition 1 *For every Datalog program P and economic policy $\boldsymbol{E}$ for P, $\boldsymbol{f} \in [P, \boldsymbol{E}](I')$ implies $\boldsymbol{f} \in [P, \boldsymbol{E}](I)$, for all $I' \subseteq I$. More specifically, if $\boldsymbol{f}$ is derived by $\boldsymbol{E}$ for I' in round i on server s, then $\boldsymbol{f}$ is derived by $\boldsymbol{E}$ for I in round $j \leq i$ on server s.*

For the proof, we first extend the concept of proof tree for Datalog programs, to annotated proof trees for Datalog evaluation with economic policies. For program P, economic policy $\boldsymbol{E}$, instance I, and fact $\boldsymbol{f}$, an *annotated proof tree* T is a proof tree

for P, I, and $\boldsymbol{f}$, where, additionally, every node $\boldsymbol{g}$ in T has a label $server_T(\boldsymbol{g})$. For non-leafs we assume the following constraint:

– $\boldsymbol{g} \in \mathsf{facts}_{\boldsymbol{P}}(server_T(\boldsymbol{g}))$, and $children_T(\boldsymbol{g}) \subseteq \mathsf{facts}_{\boldsymbol{C}}(server_T(\boldsymbol{g}))$.

We also assign to all nodes in T a number $round_T(\boldsymbol{g})$, which is obtained through the following iterative argument: For leaf nodes $\boldsymbol{g}$ in T, $round_T(\boldsymbol{g}) = 1$. For all nodes $\boldsymbol{g}$, for which all nodes in $children_T(\boldsymbol{g})$ have already a number assigned, let $max_{\boldsymbol{g}} = \max_{\boldsymbol{g}' \in children_T(\boldsymbol{g})}\{round_T(\boldsymbol{g}')\}$ and $L = \{\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k\} \subseteq children_T(\boldsymbol{g})$ be exactly those child nodes, with $round_T(\boldsymbol{g}_i) = max_{\boldsymbol{g}}$.

Now, we define $round_T(\boldsymbol{g}) = max_{\boldsymbol{g}}$ if $server_T(\boldsymbol{g}_i) = server_T(\boldsymbol{g})$, for all $\boldsymbol{g}_i \in L$, and $round_T(\boldsymbol{g}) = max_{\boldsymbol{g}} + 1$ otherwise.

Intuitively, an annotated proof tree encodes possible runs in the evaluation of P over I using $\boldsymbol{E}$. More specifically, T encodes an upperbound on the moment where a fact is derived during the evaluation. More formally:

Lemma 1 *For a Datalog program P, economic policy $\boldsymbol{E}$, instance I, and fact $\boldsymbol{f}$, the following implications hold:*

1. *$\boldsymbol{f} \in [P, \boldsymbol{E}](I)$ implies existence of an annotated proof tree T for P, $\boldsymbol{E}$, I and $\boldsymbol{f}$. If $\boldsymbol{f}$ is derived by $\boldsymbol{E}$ in round i on server s, then T exists with $round_T(root_T) = i$ and $server_T(root_T) = s$.*
2. *Existence of annotated proof tree T for $\boldsymbol{E}$, P, I and $\boldsymbol{f}$ implies $\boldsymbol{f} \in [P, \boldsymbol{E}](I)$. More specifically, $\boldsymbol{f}$ is derived on server $server_T(\boldsymbol{f})$ in round $j \leq round_T(\boldsymbol{f})$.*

Proof (1) The proof is by induction on the round in which $\boldsymbol{f}$ is derived. Clearly, after round 1, all facts residing in the network have a desired annotated proof tree. The proof then proceeds by induction, assuming that condition (1) of the lemma holds up to $\leq k$ rounds, for some k. Now suppose that $\boldsymbol{f}$ is derived at round $k+1$ on server s. The latter means that some proof-tree T for $\boldsymbol{f}$, P and $\mathsf{rec}_s^k \cup \mathsf{local}_s^{k-1}$ exists. We and set for all facts $\boldsymbol{g}$ in T, $server_T(\boldsymbol{g}) = s$. Since for all leaves $\boldsymbol{g}$ in T there exists a desired annotated proof-tree T' with $round_T[T'](\boldsymbol{g}) \leq k$ (by the hypothesis), we can simply attach these to T. It is now easy to see that $round_T(\boldsymbol{g}) \leq k+1$, for all nodes $\boldsymbol{g}$ in T'. Hence, the proof-tree is as desired.

(2) By definition of annotated proof-tree, particularly due to the constraints on $server_T$, fact $\boldsymbol{f}$ becomes derivable on server s during the computation of $\boldsymbol{E}$ over I. We only need to show that this happens in round at most $round_T(\boldsymbol{f})$. The proof is by induction on $round_T(\boldsymbol{f})$.

Clearly, if $round_T(\boldsymbol{f}) = 1$, then all facts $\boldsymbol{g}$ in T are marked $round_T(\boldsymbol{g}) = 1$, and therefore, $server_T(\boldsymbol{g}) = s$. The latter means that all leafs of T where present on node s after the first communication phase, and thus either $\boldsymbol{f} \in I$, or (because T is also a valid proof tree for $\boldsymbol{f}$), $\boldsymbol{f}$ has been derived on node s in the first computation phase.

Assume now that condition (2) holds for $i \leq k$ (induction hypothesis). Suppose $round_T(\boldsymbol{f}) = k+1$. By the induction hypothesis, all facts $\boldsymbol{g}$ in T with $round_T(\boldsymbol{f}) \leq k$ have been derived at some node in round $\leq k$. Now it is easy to see that the top-

fragment T' of T (i.e., all the subtree of all facts marked with $round_T(\boldsymbol{g}) = k + 1$ and their immediate children), describes a proof-tree for $\boldsymbol{f}$ and P on node s.

Let j be the earliest communication round after which all leaf nodes have reached server s. Since all leaf-nodes have been derived in round $\leq k$ (by the hypothesis), and the semantics of $\boldsymbol{E}$ and T guarantees arrival of these facts on server s after the next communication phase, we have that $j \leq k + 1$. It is now easy to see (due to T'), that $\boldsymbol{f}$ will be derived on server s in computation round $j \leq k + 1$. This concludes the proof. □

Since for instances $I' \subseteq I$, every annotated proof tree T for $\boldsymbol{E}$, P, and I' is trivially also an annotated proof tree for $\boldsymbol{E}$, P, and I, Proposition 1 is now a corollary of Lemma 1.

5 Parallel-Correctness

An economic policy for a Datalog program does not necessarily lead to the desired output. For example, if the production policy maps every fact onto the empty set of servers, then the execution will generate only empty intensional database relations. Henceforth, we are only interested in economic policies that generate the expected output.

Definition 3 (**Parallel-correctness**) An economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ is *parallel-correct* for Datalog program P if $[P, \boldsymbol{E}](I) = P(I)|_{out(P)}$, for every instance $I \subseteq \mathsf{facts}(\text{EDB}(P))$.

Parallel-correctness is in general undecidable, even for simple classes of policies. For instance, consider the class of policies, where $\boldsymbol{P}(\boldsymbol{f}_1) = \boldsymbol{P}(\boldsymbol{f}_2)$ and $\boldsymbol{C}(\boldsymbol{f}_1) = \boldsymbol{C}(\boldsymbol{f}_2)$, whenever $\boldsymbol{f}_1, \boldsymbol{f}_2$ are facts with the same relation symbol. We call this class of policies *value-independent*, denoted $\mathcal{E}_{indep}$, since the facts are mapped to servers only according to the relations they belong to. Value-independent policies allow a succinct representation by simply enumerating the intensional relation names of P and the subsets of $[p]$ where each relation is assigned.

We consider the following decision problem.

	$\text{PC}(Datalog, \mathcal{E}_{indep})$
Input:	Datalog program P, policy $\boldsymbol{E} \in \mathcal{E}_{indep}$.
Question:	Is $\boldsymbol{E}$ parallel-correct for P?

Theorem 1 $\text{PC}(Datalog, \mathcal{E}_{indep})$ *is undecidable.*

Proof The proof is by a reduction from the Datalog containment problem, which is well-known to be undecidable [1]. Let P_1 and P_2 be two arbitrary Datalog programs given as input for the containment problem. We assume that both are over the same nullary output relation name, say O.

We first denote by P_i^* an indexed version of program P_i; particularly we define P_i^* as P_i in which all intensional relation names are annotated with index i. We now construct a program P by taking all rules from P_1^* and P_2^*, and adding the rules $O() \leftarrow O_i()$, for $i \in \{1, 2\}$. We note that $edb(P) = edb(P_1^*) \cup edb(P_2^*)$ and $out(P) = \{O\}$. As economic policy we take $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ over the 2-node network $\{1, 2\}$. The consumption policy maps all facts with index i to server i. The production policy maps all facts with index i to server i, and the fact $O()$ to server 2. The extensional database facts are consumed on all servers.

Intuitively, programs P_1^* and P_2^* are computed locally on server 1 and server 2. It thus follows from the construction that (†) $P_1(I) \cup P_2(I) \subseteq [P, \boldsymbol{E}](I)$, for every instance I. Notice that rule $O() \leftarrow O_1()$ is never used, since server 2 cannot consume facts over relation names with index 1.

It remains to show that $\boldsymbol{E}$ is parallel-correct for P if and only if $P_1 \subseteq P_2$. Indeed, if $P_1 \subseteq P_2$, then $P(I) = P_2(I)$ for every instance I, which implies that the policy will compute the correct result for O. The other direction follows from monotonicity of P. From (†) it follows that this condition is satisfied if and only if all facts over the O relation produced by $P(I)$ are also produced by $[P, \boldsymbol{E}](I)$, which is the case only if $O() \in P(I)$ implies $O_2() \in P(I)$. The latter is equivalent to saying $O() \in P_1(I)$ implies $O() \in P_2(I)$ for every instance I, which means that $P_1 \subseteq P_2$. □

In fact, the above proof yields an even stronger result:

Lemma 2 *Let P be an arbitrary Datalog program and $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ an economic policy over σ that is parallel-correct for P. Let $\boldsymbol{f} \in \mathsf{facts}(\sigma)$ and $\boldsymbol{C}'$ be a consumption policy with $\boldsymbol{C}'(\boldsymbol{g}) = \boldsymbol{C}(\boldsymbol{g})$ for all $\boldsymbol{g} \in \mathsf{facts}(\sigma) \setminus \{\boldsymbol{f}\}$ and $\boldsymbol{C}'(\boldsymbol{f}) \subsetneq \boldsymbol{C}(\boldsymbol{f})$. It is undecidable whether $\boldsymbol{E}' = (\boldsymbol{P}, \boldsymbol{C}')$ is parallel-correct for P.*

Proof We simply observe that the economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ in the proof of Proposition 1 has this property. Indeed, updating $\boldsymbol{C}(O_1()) = \{1\}$ to $\boldsymbol{C}(O_1()) = \{1, 2\}$ makes the policy trivially parallel-correct for P. □

Despite the above results, we can present some syntactic conditions that are necessary for parallel-correctness, and some that are sufficient.

Definition 4 (**Support**) An instantiation $v(\tau)$ of rule τ with valuation v is *supported* by economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ if there exists some server $s \in [p]$ with $v(head_\tau) \in \mathsf{facts}_{\boldsymbol{P}}(s)$ and $v(body_\tau) \subseteq \mathsf{facts}_{\boldsymbol{C}}(s)$.

We say that an economic policy $\boldsymbol{E}$ *supports a proof tree T* if all the rule instantiations in T are supported.

Lemma 3 *Let P be a Datalog program and $\boldsymbol{E}$ an economic policy. If a proof tree T for P is supported by $\boldsymbol{E}$, then for every instance I, with $fringe_T \subseteq I$, we have $root_T \in [P, \boldsymbol{E}]$.*

Proof The proof is by induction on the depth d of T. Particularly we show using a simple inductive argument that $root_T \in \mathsf{local}_i^k$, for some server i and $k \leq d$, which implies $root_T \in [P, \boldsymbol{E}]$. Recall that local_i^k denotes the facts residing locally on server i after the k-th computation round.

As base case let $d = 1$, meaning that T describes a single rule instantiation. After the first communication round, all servers j have $\mathsf{local}_j^0 \cup \mathsf{rec}_j^1 \subseteq I \cap \mathsf{facts}_{\boldsymbol{C}}(j)$. By the assumption that $\boldsymbol{E}$ supports T, it follows that

$$children_T(root_T) \subseteq \mathsf{facts}_{\boldsymbol{C}}(i) \cap I \subseteq \mathsf{local}_i^1; \text{ and}$$
$$root_T \in \mathsf{facts}_{\boldsymbol{P}}(i),$$

for some server i, thus after the first computation round, $root_T \in \mathsf{local}_i^1$.

For $d > 1$ we observe that $root_T$ and its children in T define a rule instantiation (τ, v), and, by the assumptions of the lemma, this rule instantiation is supported by $\boldsymbol{E}$. More specifically, some server i exists where $root_T \in \mathsf{facts}_{\boldsymbol{P}}(i)$ and $children_T(root_T) \subseteq \mathsf{facts}_{\boldsymbol{C}}(i)$. Further, for all facts $\boldsymbol{f} \in children_T(root_T)$, the respective subtree $T_{\boldsymbol{f}}$ of T with root $\boldsymbol{f}$ is supported by $\boldsymbol{E}$ and with depth $d - 1$. By the induction hypothesis it follows that for all these facts $\boldsymbol{f}$ there is a server j and $k \leq d - 1$, where $\boldsymbol{f} \in \mathsf{local}_j^k$. Therefore $children_T(root_T) \subseteq \mathsf{local}_i^{k^*} \cup \mathsf{rec}_i^{k^*}$, where k^* denotes the maximal k, and consequently, $root_T \in \mathsf{local}_i^{k^*+1} \subseteq \mathsf{local}_i^d$. □

We now have a characterization for parallel-correctness of a program P w.r.t. an economic policy.

Proposition 2 *Let P be a Datalog program. An economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ is parallel-correct for P if and only if for every proof tree for P with fringe over $\mathsf{facts}(\sigma(P))$, a subsumed supported proof tree exists.*

Proof (If) Let I be an arbitrary instance, we show $P(I) = [P, \boldsymbol{E}](I)$. By monotonicity, $[P, \boldsymbol{E}](I) \subseteq P(I)$, thus we focus on completeness. For this, let $\boldsymbol{f} \in P(I)$, which means that a proof tree T exists with $fringe_T \subseteq I$ and $root_T = \boldsymbol{f}$. Particularly, by the assumption of the lemma we can choose T so that it is also supported by $\boldsymbol{E}$. It now follows from Lemma 3 that $\boldsymbol{f} \in [P, \boldsymbol{E}]$.

(Only if) We assume $(\boldsymbol{P}, \boldsymbol{C})$ is parallel-correct for P. Let T be an arbitrary proof tree. The proof is by construction following the derivation of $root_T$ using $\boldsymbol{E}$. First, from parallel-correctness it follows that $P(I) = [P, \boldsymbol{E}](I)$, for any instance I. Here we take $I = fringe_T$, implying $root_T \in [P, \boldsymbol{E}](I)$. The proof now continues by induction on the number of rounds needed for $\boldsymbol{E}$ to derive $root_T$.

The induction hypothesis is that if k rounds are needed to derive $root_T$, then a supported proof-tree of depth k subsumed by T exists.

As a base case suppose $k = 1$. That is, $root_T \in \mathsf{local}_i^1$, meaning that $root_T \in P_{\restriction \boldsymbol{E}}(\mathsf{local}_j^0 \cup \bigcup_j \mathsf{rec}_j^1)$ for some server j. Particularly, a valuation v and rule $\tau \in P$ existed with $v(body_\tau) \subseteq \mathsf{facts}_{\boldsymbol{C}}(j) \cap I$ and $v(head_\tau) = root_T$, which means that the corresponding rule instantiation is supported by $\boldsymbol{E}$. Here, the proof tree admitted by (τ, v) is as desired.

For $k > 1$ the proof is analogous, but now we take as proof tree the tree obtained by concatenating the rule instantiation with the proof trees for each child. Existence of

the latter follows from the induction hypothesis. As the number of rounds decreases by one in each inductive step, and the fringes of the obtained trees cannot have other facts than does in I, the constructed proof tree is as again as desired. □

We consider various categories of economic policies based on which rule instantiations are supported for a given Datalog program P:

$\mathcal{N}_P^{all}$: the set of all rule instantiations of P.
$\mathcal{N}_P^{min}$: the set of all minimal rule instantiations of P. An instantiation of rule τ with valuation v is *minimal* if there is no rule τ' and valuation v' with $v'(head_{\tau'}) = v(head_\tau)$ and $v'(body_{\tau'}) \subsetneq v(body_\tau)$.
$\mathcal{N}_P^{use}$: the set of all rule instantiations of P that are useful. Recall that an instantiation of rule τ with valuation v is useful if $v(head_\tau) \notin v(body_\tau)$.
$\mathcal{N}_P^{ess}$: the set of all essential rule instantiations of P. An instantiation of rule τ with valuation v is *essential* if for some P-derivable fact $\boldsymbol{f}$ and instance I, every proof tree T for $\boldsymbol{f}$ on I and P has a vertex $\boldsymbol{g}$ with $\boldsymbol{g} = v(head_\tau)$ and $v(body_\tau) \subseteq children_T(\boldsymbol{g})$.

If the program is non-recursive, then $\mathcal{N}_P^{use} = \mathcal{N}_P^{all}$, since there will be no rule that contains the same relation in the head and the body. We also have:

Proposition 3 *For every Datalog program P, we have $\mathcal{N}_P^{ess} \subseteq \mathcal{N}_P^{min} \cap \mathcal{N}_P^{use}$.*

Before giving a proof, we first show the following Lemma.

Lemma 4 *For every proof tree T of depth d, there exists a proof tree $T' \sqsubseteq T$ of depth at most d that uses only minimal and useful rule instantiations.*

Proof The proof is by induction on the depth of T, which we denote d.

For the base case, let $d = 1$. Then, T corresponds to a single rule instantiation (τ, v) for P where all the facts in $v(body_\tau)$ are extensional database facts. By definition, there is also a minimal rule instantiation (τ', v'), with $v'(head_{\tau'}) = v(head_\tau)$ and $v'(head_{\tau'}) \subseteq v(body_\tau)$, which admits the desired proof tree.

As induction hypothesis we take the statement of the lemma. Now, for the induction step, suppose T has depth $d > 1$. Then, the root of T, together with its children, defines a rule instantiation (τ, v) for P. Now take an subsumed minimal instantiation (τ', v), such that $v'(head_{\tau'}) = v(head_\tau)$ and $v'(body_{\tau'}) \subseteq v(body_\tau)$. For every fact $\boldsymbol{f} \in v'(head_{\tau'})$, let $T_{\boldsymbol{f}}$ be the subtree of T with root $\boldsymbol{f}$ (child of $root_T$). By the induction hypothesis, there is a proof tree $T'_{\boldsymbol{f}} \sqsubseteq T_{\boldsymbol{f}}$ with depth $\leq d-1$ that uses only minimal rule instantiations. The proof tree that combines instantiation (τ', v') with $T'_{\boldsymbol{f}}$ for all $\boldsymbol{f} \in v'(\tau')$ is as desired. □

Proof of Proposition 3 The containment $\mathcal{N}_P^{ess} \subseteq \mathcal{N}_P^{use}$ is straightforward, since a proof tree does not use any useless rule instantiations. We next show that $\mathcal{N}_P^{ess} \subseteq \mathcal{N}_P^{min}$. Suppose that we have an instantiation of rule τ with valuation v that is essential. Then, there exists some fact $\boldsymbol{f}$ and instance I for which every proof tree T

has a vertex $\boldsymbol{g}$ with $\boldsymbol{g} = v(head_\tau)$ and $v(body_\tau) \subseteq children_T(\boldsymbol{g})$. By Lemma 4, we can pick this tree such that it uses only minimal rule instantiations. This implies that the rule instantiation with head $\boldsymbol{g}$ and body $children_T(\boldsymbol{g})$ is minimal. Hence, the instantiation with head $v(head_\tau)$ and body $v(body_\tau)$ is also minimal. □

The following example demonstrates the different types of rule instantiations.

Example 2 Let P be the left-linear transitive closure program from Example 1; consider a rule instantiation of the recursive rule: $T(a, b) \leftarrow T(a, c), R(c, b)$, for some (not necessarily different) constants a, b, c. We distinguish the following cases:

$c = a$: in this case, the instantiation is not minimal, since we can derive the same head fact from the instantiation $T(a, b) \leftarrow R(a, b)$ of the first rule.

$c = b$: in this case, the instantiation is useless, since $T(a, b)$ also belongs in the body.

$c \neq a, c \neq b$: in this case, the instantiation is minimal and useful; it is also essential. To show this, consider the instance $I = \{R(a, c), R(c, b)\}$, and the fact $\boldsymbol{f} = T(a, b)$. Because $c \notin \{a, b\}$, the only proof tree for $\boldsymbol{f}$ without "useless" rule instantiations is the one with root $\boldsymbol{f}$, children $T(a, c)$, $R(c, b)$, where $T(a, c)$ has $R(a, c)$ as child.

Depending on which types of rule instantiations are supported by an economic policy, we can define different types of policies. An economic policy that supports all possible rule instantiations, that is, $\mathcal{N}_P^{all}$, is said to be *strongly supporting* for Datalog program P.

Proposition 4 *Let P be a Datalog program and $\boldsymbol{E}$ an economic policy. If $\boldsymbol{E}$ supports all minimal and useful rule instantiations in P, then it is parallel-correct. If $\boldsymbol{E}$ is parallel-correct for P, then it supports all essential rule instantiations.*

Proof The first item follows from Proposition 2 and Lemma 4. For the second item, consider a parallel-correct policy $\boldsymbol{E}$ and an essential instantiation of rule τ with valuation v. By the definition of essential, for some fact $\boldsymbol{f}$ and instance I, every proof tree T for $\boldsymbol{f}$ on I and P has a vertex $\boldsymbol{g}$ with $\boldsymbol{g} = v(head_\tau)$ and $v(body_\tau) \subseteq children_T(\boldsymbol{g})$. By Proposition 2, there must exist such a tree T that is supported. This implies that there exists server s with $v(head_\tau) = \boldsymbol{g} \in \mathsf{facts}_{\boldsymbol{P}}(s)$ and $v(body_\tau) \subseteq children_T(\boldsymbol{g}) \subseteq \mathsf{facts}_{\boldsymbol{C}}(s)$. Hence, the essential rule instantiation is indeed supported. □

Proposition 5 *Let P be a Datalog program where each intensional relation name occurs only in the head of rules (i.e., P is a union of CQs). Then, $\mathcal{N}_P^{ess} = \mathcal{N}_P^{min} \cap \mathcal{N}_P^{use}$.*

Proof Because P is not recursive, $\mathcal{N}_P^{use} = \mathcal{N}_P^{all}$; hence, because of Proposition 3 it suffices to show that $\mathcal{N}_P^{min} \subseteq \mathcal{N}_P^{ess}$. Indeed, consider a minimal instantiation for rule

τ with valuation v, and consider the instance $I = v(body_\tau)$ and fact $\boldsymbol{f} = v(head_\tau)$. Take any proof tree T for $\boldsymbol{f}$ on I and P; T must have depth one. Because of the minimality of the rule instantiation, it must be that $children_T(\boldsymbol{f}) = v(body_\tau)$, which proves the essentiality. □

Together with Proposition 4, the above proposition implies that a Datalog program where the body of each rule contains only extensional database relations is parallel-correct if and only if it supports every minimal rule instantiation, or equivalently if and only if it supports every essential rule instantiation. Notice that this class of Datalog programs corresponds to a program that computes a set of UCQs, and thus the above result captures the characterization of parallel-correctness for CQs and UCQs in [7, 17]. We should emphasize here that [7, 17] consider only economic policies where $\boldsymbol{P}$ assigns every fact to every server, while a general economic policy can assign facts to any subset of servers.

For general Datalog programs, $\mathcal{N}_P^{ess} = \mathcal{N}_P^{min} \cap \mathcal{N}_P^{use}$ is not true anymore, and thus supporting essential instantiations is not a sufficient condition for parallel-correctness, even if P is non-recursive. (Recall that non-recursiveness is a syntactic condition, and that all such programs are straightforwardly rewritable to UCQs.)

Example 3 Consider the following non-recursive Datalog program P:

$$V() \leftarrow R(x, y), S(z, w), S(w, z). \qquad U() \leftarrow V(), R(x, y), S(z, w).$$

and take the rule instantiation with head $U()$ and body $\{V(), R(a, b), S(c, d)\}$. Assume that $c \neq d$. This rule instantiation is minimal, but we will show that it is not essential.

For the sake of contradiction, assume that it is essential. Then, for some instance I there exists a proof tree T for $U()$ on I and P such that there exists a vertex $U()$ with $\{V(), R(a, b), S(c, d)\} \subseteq children_T(U())$. Since the proof tree contains the fact $V()$, it also contains a rule instantiation that derives the fact $V()$ with body $\{R(a', b'), S(c', d'), S(d', c')\}$ for some constants a', b', c', d'. We can now construct two proof trees for $U()$ on the same instance, as seen in Fig. 1. Because $c \neq d$, one of the facts $S(c', d'), S(d', c')$ must be different from $S(c, d)$ (In Fig. 1 we assume this fact is $S(d', c')$). Thus, for one of the two trees, the children of $U()$ will not be a subset of $\{V(), R(a, b), S(c, d)\}$. This implies that the rule instantiation we considered is indeed not essential.

Example 4 This example shows that $\mathcal{N}_P^{ess} = \mathcal{N}_P^{min} \cap \mathcal{N}_P^{use}$ can hold for recursive programs. Consider Example 2. Notice that every rule instantiation of the base rule,

Fig. 1 The two proof trees for the fact $\boldsymbol{f} = U(a, b)$

$T(x, y) \leftarrow R(x, y)$, is trivially minimal, useful and essential. As for the recursive rule, we showed in Example 2 that an instantiation that is minimal and useful is also essential. Observe that if this instantiation is only minimal but not useful, or only useful and not minimal, it is not essential. Thus, both properties are necessary to guarantee essentiality.

We conclude this section by commenting on whether it is computationally feasible to test the different properties of rule instantiations. It is easy to see that given an instantiation, it is possible to check whether it is useful in polynomial time. The complexity for checking the minimality of a rule instantiation is coNP-complete [7]. Unfortunately, testing essentiality of a rule instantiation is undecidable.

Proposition 6 *Testing essentiality of rule instantiations, as well as whether for a given rule an essential instantiation exists, is undecidable.*

Proof We first show the latter. The proof is again by a reduction from the Datalog containment problem. For this let P_1 and P_2 be programs serving as input, with output predicate $O^{(k)}$. As before, let P_1^* and P_2^* be the indexed versions of these programs.

Define Datalog program P, with $edb(P) = edb(P_1^*) \cup edb(P_2^*)$, $out(P) = \{O^{(k)}\}$, and $idb(P) = idb(P_1^*) \cup idb(P_2^*) \cup out(P)$. Program P is defined as the union of P_1^*, P_2^*, and the following rules.

$$O(\boldsymbol{x}) \leftarrow O_1(\boldsymbol{x}).$$
$$O(\boldsymbol{x}) \leftarrow O_2(\boldsymbol{x}).$$

Now the question whether $P_1 \subseteq P_2$ reduces to the question whether some essential rule instantiation for $O(\boldsymbol{x}) \leftarrow O_1(\boldsymbol{x})$ exists. Indeed, if $P_1 \subseteq P_2$, this cannot be the case, since a proof tree over $\{O\} \cup \sigma P_2$ will always exist.

If $P_1 \not\subseteq P_2$, then some I and $\boldsymbol{t}$ exist, with $O_1(\boldsymbol{t}) \in P_1(I)$, $O_2(\boldsymbol{t}) \notin P_2(I)$. Then, it is easy to see that all proof trees T with $root_T = O(\boldsymbol{t})$ contain the instantiation $O(\boldsymbol{t}) \leftarrow O_1(\boldsymbol{t})$, which is thus essential.

Next, we show that essentiality of rule instantiations is undecidable. More precisely the proof is by contradiction. We show that, if essentiality of rule instantiations is decidable, then testing whether an essential instantiation for a given rule exists is decidable as well, which contradicts the earlier obtained result.

The algorithm relies on the observation that positive Datalog programs (without function symbols) are C-generic, with C being the constants occurring in P. Thus, if a rule instantiation is essential, all isomorphic instantiations (where values from C are preserved) are essential. Clearly there are only finitely many distinct instantiations (up to isomorphisms).

For given rule $\tau \in P$, one can thus iterate over the above defined equivalence class, choose from each a specific instantiation, and test whether the chosen instantiation is essential. An essential instantiation is found if and only if the rule has an essential instantiation. □

6 Generalized Hypercube Policies

In this section, we present a general class of economic policies, called *Generalized Hypercube Policies (GHP)*, which encompass a broad variety of evaluation strategies.

We first give an intuitive explanation. The formalism of GHPs relies on the Hypercube partitioning for CQs [4], which has been shown to provide guarantees on the communication-cost for CQ evaluation [9]. Let $P = \{\tau\}$ be a CQ with k distinct variables. Hypercube conceptually orders the p servers as a hypercube $\boldsymbol{H} = [p_1] \times [p_2] \times \cdots \times [p_k]$, with $\prod_i p_i = p$, where every dimension $p_i \geq 0$ corresponds to a variable x_i from the query; every server is assigned a unique point in the space $\boldsymbol{H}$; and every variable x_i is associated to a hash function $h_{x_i} : \mathbf{dom} \mapsto [p_i]$. Then, a fact $R(a_1, \ldots, a_r)$, matching with atom $R(y_1, \ldots, y_r) \in body_\tau$, is sent to all servers whose coordinate in the dimension of $\boldsymbol{H}$ associated to variable y_i is equal to $h_{y_i}(a_j)$, for all $j \in [r]$. Then, program P is computed on each server over the data at hand.

For GHPs, we associate to *every* rule a hypercube over the full p-server network, and intuitively define the consumption policy so that "a fact is consumed at server i if and only if one of the considered Hypercube specifications would send it to server i", and "a fact is in the production policy of server i if and only if one of the Hypercube specifications would derive it on server i".

GHP Parameters Let P be a Datalog program, and assume we have a network $[p]$. A *GHP for* P defines a finite set of k-dimensional *hypercubes* $\boldsymbol{H}^1, \ldots, \boldsymbol{H}^\ell$, for some parameter k. We note that the assumption that every hypercube has the same number of dimensions is without loss of generality, since we allow the range [1] for dimensions. The range of the dimensions of the hypercubes are parametrized by a matrix of dimensions $\ell \times k$ with entries $p_{i,j}$, such that $\prod_{i=1}^{k} p_{j,i} = p$, for each $j \in [\ell]$. Each hypercube is then defined as $\boldsymbol{H}^j = [p_{j,1}] \times [p_{j,2}] \times \ldots [p_{j,k}]$. For each hypercube $\boldsymbol{H}^j$, we also define a bijective mapping map^j that assigns to every point in $\boldsymbol{H}^j$ a server $s \in [p]$. The latter thus provides the mapping between conceptual servers in the cube and real servers in the considered network.

A GHP policy next assigns each rule τ to exactly one of the hypercubes: let $\chi : P \to [\ell]$ be the function that encodes this assignment. Given this assignment, a GHP defines a mapping $\rho^\tau : [k] \to \mathcal{P}(vars(\tau))$ that maps each dimension of the hypercube $\boldsymbol{H}^{\chi(\tau)}$ to a subset of the variables that appear in τ.

Finally, the GHP defines for each dimension $i \in [k]$ and each hypercube $\boldsymbol{H}^j$ a *hash function* h_i^j that maps subsets[3] of **dom** with size lesser or equal than the largest size of $\rho^\tau(i)$ (for any τ with $\chi(\tau) = j$) to a value in the i-th dimension. For hash functions that accept non-empty sets, we require that they are surjective. Notice that our concept of hash-function is a generalization of the hash-functions used in, e.g., the Hypercube algorithm, where $\alpha = 1$. This generalization allows to scatter tuples over a row in a more fine-grained way than is possible via a single variable. Further,

[3]We define hash functions over sets rather than tuples, because sets naturally enforce order invariance, which is a property that we rely on later in the paper.

we notice that, by definition, rules that use the same hypercube, also use the same hash function for each dimension of that hypercube.

GHP Semantics Let $\boldsymbol{f}$ be a fact and suppose that $\boldsymbol{f} = v(A)$, for some valuation v and atom $A = R(\boldsymbol{y})$ that appears in rule τ.[4] We define the following set of servers:

$$S^{\tau}_{\boldsymbol{f},A} = \Big\{ \mathsf{map}^{\chi(\tau)}(\mathbf{q}) \mid \mathbf{q} \in \boldsymbol{H}^{\chi(\tau)} \text{ such that } \forall \ i \text{ with } \emptyset \subsetneq \rho^{\tau}(i) \subseteq \boldsymbol{y} : \mathbf{q}_i = h_i^{\chi(\tau)}(v(\rho^{\tau}(i))) \Big\}.$$

Intuitively, $S^{\tau}_{\boldsymbol{f},A}$ denotes the set of servers whose coordinate $\boldsymbol{q}$ is consistent with the hash mappings specified for τ. Notice that if the atom $R(\boldsymbol{y})$ has only a part of the variables that correspond to some dimension i, then facts are broadcast over dimension i, as it happens if none of these variables are in $\boldsymbol{y}$.

The consumption policy $\boldsymbol{C}(\boldsymbol{f})$ is defined as the union over all sets $S^{\tau}_{\boldsymbol{f},A}$ for rules τ and atoms $A \in \mathit{body}_{\tau}$ with instantiation $\boldsymbol{f}$. The production policy $\boldsymbol{P}(\boldsymbol{f})$ is similarly defined as the union over all sets $S^{\tau}_{\boldsymbol{f},A}$ for rules τ and atom head_{τ} with instantiation $\boldsymbol{f}$.

Example 5 Consider the Datalog program depicted in Fig. 2. We choose two hypercubes $\boldsymbol{H}^1$, $\boldsymbol{H}^2$ ($\ell = 2$) with dimension $k = 2$. The first two rules τ_1, τ_2 are mapped to the hypercube $\boldsymbol{H}^1$, and the third rule τ_3 is mapped to $\boldsymbol{H}^2$. We choose the dimensions of the hypercubes such that $p_{1,1} \cdot p_{1,2} = p$, $p_{2,1} = p$, and $p_{2,2} = 1$. The two functions map^1, map^2 map the points of $\boldsymbol{H}^1$, $\boldsymbol{H}^2$ respectively to $\{1, \ldots, p\}$ in a one-to-one fashion. Finally, the mapping of variables to dimensions is:

$$\rho^{\tau_1}(1) = \{x\}, \rho^{\tau_1}(2) = \{y\},$$
$$\rho^{\tau_2}(1) = \{x\}, \rho^{\tau_2}(2) = \{z\},$$
$$\rho^{\tau_3}(1) = \{y\}, \rho^{\tau_3}(2) = \{\}.$$

Consider the first two rules (which form the left-linear TC example), and assume that $p_{1,1} = 1$ and $p_{1,2} = p$. Then, the resulting GHP is equivalent to the hash partitioning policy that we described in Example 1. Notice that since we use the same hypercube for both rules, the extensional database relation R will be hash partitioned only once. If we now change the dimensions to $p_{1,1} = p$, $p_{1,2} = 1$, we obtain the decomposable policy of Example 1 that broadcasts the extensional R to every server and can terminate in a single round. Apart from the above two GHPs, we can also define other GHPs by configuring different dimensions of the hypercube $\boldsymbol{H}^1$. For example, we can choose $p_{1,1} = p_{1,2} = \sqrt{p}$.

We next show that GHPs are strongly supporting policies.

Proposition 7 *Let P be a Datalog program. Every GHP $\boldsymbol{E}$ for P is strongly supporting for P and, as a consequence, parallel-correct for P.*

[4]Notice that either v does not exist, or is unique for the variables in atom A.

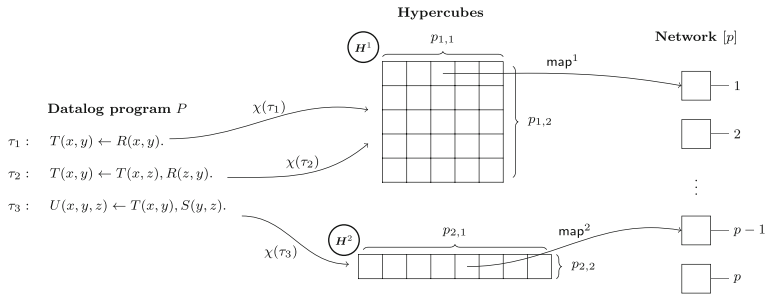

Fig. 2 Example of a GHP policy for the Datalog program P with three rules

Proof To show that $\boldsymbol{E}$ is strongly supporting, consider some rule $\tau \in P$, and its instantiation w.r.t. some valuation v. Consider some atom $A = R(\boldsymbol{y})$ in the body of τ; then the consumption policy says that its instantiation $\boldsymbol{f} = v(A)$ will be consumed in the set $S^{\tau}_{\boldsymbol{f},A}$. Similarly if A is the head, the fact $\boldsymbol{f}$ will be produced in $S^{\tau}_{\boldsymbol{f},A}$. Now we can write the intersection $\bigcap_{A\in\tau} S^{\tau}_{\boldsymbol{f},A}$ as:

$$\bigcap_{A\in\tau} \{\mathsf{map}^{\chi(\tau)}(\mathbf{q}) \mid \forall i : \emptyset \subsetneq \rho^{\tau}(i) \subseteq vars(A) \Rightarrow \mathbf{q}_i = h_i^{\chi(\tau)}(v(\rho^{\tau}(i)))\}$$
$$\supseteq \{\mathsf{map}^{\chi(\tau)}(\mathbf{q}) \mid \forall i : \mathbf{q}_i = h_i^{\chi(\tau)}(v(\rho^{\tau}(i)))\} \supsetneq \emptyset$$

In other words, there will be at least one server in $\bigcap_{A\in\tau} S^{\tau}_{A}$, which means that every instantiation of the rule τ will be strongly supported. □

GHP Families Since we do not want to consider an encoding mechanism for hash functions—which is necessary to formally reason about properties for GHPs—we introduce the concept of GHP families. Given a Datalog program P and network $[p]$, a *GHP family* $\mathcal{F}$ is defined as the set of GHPs over P and $[p]$ that all have the same parametrization for $\mathbf{P}$, map^j, χ, ρ^{τ}. In other words, policies in $\mathcal{F}$ can differ only with respect to the choice of hash functions, and for every choice of hash functions, the associated GHP is in the family. By $\mathfrak{F}_{\mathrm{GHP}}$ we denote the class of all GHP families.

7 Bounded & Disjoint Evaluation

In this section, we we ask two main questions: First, can we reason about the number of rounds that an economic policy needs to compute a Datalog program? Second, can we constrain the number of servers that derive a copy of the same fact? We start with a formal definition of boundedness.

Definition 5 (**Boundedness**) An economic policy $\boldsymbol{E}$ for Datalog program P is *bounded* if some constant k exists such that, for every instance I, the network reaches

a global fixpoint for $\boldsymbol{E}$ and P, when round k is finished. We say $\boldsymbol{E}$ is ℓ-bounded if $k \leq \ell$.

First, we remark that setting ρ^{τ} to map to the emptyset for all rules τ, does not eliminate communication. Indeed, economic policies always send facts to all servers that may need the fact according to the policy, independently of whether the fact is already known by the target server. In other words, the responsibility to decide whether a fact is send lies entirely on the sender. There is no trivial way to provide a 1-bounded economic policy.

Second, one should not confuse the number of rounds in the parallel computation with the number of iterations of semi-naive evaluation. Nevertheless, as the following proposition shows, boundedness of the Datalog program implies boundedness of the evaluation.

Proposition 8 *If P is a bounded Datalog program, then every parallel-correct economic policy $\boldsymbol{E}$ for P is k-bounded, for some constant k that depends on P.*

Proof We use the following claim.

(†) if we run $\boldsymbol{E}$ on an instance with bounded size, then $\boldsymbol{E}$ will finish its evaluation in a bounded number of rounds.

The result now follows from boundedness of P and Proposition 1. Boundedness of P implies that some constant exists, such that for every instance I and fact $\boldsymbol{f}$, $\boldsymbol{f} \in P(I)$ implies the existence of a proof tree with depth no more than the bound. We observe that a bound on depth implies also a bound on fringe size.

Now, for arbitrary $\boldsymbol{f}$ and I, for $\boldsymbol{f} \in [P, \boldsymbol{E}](I)$ we observe that $\boldsymbol{f} \in P(I)$, due to parallel-correctness, and thus, due to boundedness of P, that some proof-tree T with bounded fringe exists. Then, it follows from (†) that the number of rounds of $\boldsymbol{E}$ on the instance consisting only of the fringe is bounded, and due to parallel-correctness of $\boldsymbol{E}$, that $\boldsymbol{f} \in [P, \boldsymbol{E}](\mathit{fringe}_T)$.

Since this observation holds for all $\boldsymbol{f} \in [P, \boldsymbol{E}](I)$, it follows from Proposition 1 that the number of rounds of $\boldsymbol{E}$ over the whole instance is also bounded.

It remains to show (†). The crucial observation is that, in all but the last computation round at least some fact is communicated in the network that has not been communicated in any earlier round. Indeed, only new derivations can trigger a next communication round, and when a fact is received it will trigger new derivations only if it is not already known by the receiving server.

Since the instance is bounded, the active domain (of this instance) is bounded, and thus the number of facts that can be introduced during the evaluation is bounded as well. The result follows. □

Surprisingly, there exist economic policies for bounded Datalog programs that are not bounded. However, due to Proposition 8, such policies cannot be parallel-correct.

Example 6 Consider the following bounded program.

$$T(x) \leftarrow A(x). \qquad T(x) \leftarrow B(x), T(y).$$

We construct a network with $p > 1$ servers. Consider a policy that consumes $T(i)$ and $B(i)$ at server $(i \mod p) + 1$, and produces $T(i)$ at server $(i \mod p)$. Every tuple in A is consumed at server 1. Now, consider the following input instance: $\{A(0), B(1), B(2), \ldots, B(p-1)\}$. It is easy to see that $T(0)$ is produced at server 1 at round 1, $T(1)$ is produced at server 2 at round 2, and so on, until $T(p-1)$ is produced at round p at server p.

In the remainder of this section, we focus on pure Datalog (*PureDatalog*). A Datalog program is *pure* if it is free of constants and variables occur at most once in every atom [28]. We emphasize that this definition prohibits a variable from occurring on multiple positions in an atom, but that a variable can still occur in multiple (distinct) atoms of a rule. We note that, for a program P in pure Datalog, every fact over a P-consumable (P-derivable) relation itself is P-consumable (P-derivable).

We consider the following decision problems.

	k-BOUNDEDNESS$(\mathcal{L}, \mathcal{E})$
Input:	Program $P \in \mathcal{L}$, policy $\boldsymbol{E} \in \mathcal{E}$.
Question:	Is $\boldsymbol{E}$ k-bounded for P?

	BOUNDEDNESS$_F(\mathcal{L}, \mathfrak{F})$
Input:	Program $P \in \mathcal{L}$, family $\mathcal{F} \in \mathfrak{F}$.
Question:	Is there a k s.t. $\mathcal{F}$ is k-bounded for P?

	k-BOUNDEDNESS$_F(\mathcal{L}, \mathfrak{F})$
Input:	Program $P \in \mathcal{L}$, family $\mathcal{F} \in \mathfrak{F}$.
Question:	Is $\mathcal{F}$ k-bounded for P?

Theorem 2 1. BOUNDEDNESS$_F$(*PureDatalog*, $\mathfrak{F}_{\text{GHP}}$) *is undecidable;*

2. k-BOUNDEDNESS(*PureDatalog*, $\mathcal{E}_{indep}$) *is undecidable for* $k \geq 2$*;*
3. k-BOUNDEDNESS$_F$(*PureDatalog*, $\mathfrak{F}_{\text{GHP}}$) *is undecidable for* $k \geq 2$*; and*
4. k-BOUNDEDNESS$_F$(*PureDatalog*, $\mathfrak{F}_{\text{GHP}}$) *is in* PTIME *if* $k = 1$.

We first show results (1), (2), and (3) (Proposition 9 and Proposition 10).

Proposition 9 k-BOUNDEDNESS$_F$(*PureDatalog*, $\mathfrak{F}_{\text{GHP}}$) *is undecidable if* $k \geq 2$. *Since the proof is by reduction to an economic policy that is either 2-bounded or not bounded et all, it follows that* BOUNDEDNESS$_F$(*PureDatalog*, $\mathfrak{F}_{\text{GHP}}$) *is undecidable.*

Proof The proof is by a reduction from the undecidable containment problem for Datalog programs. Let P_1, P_2 be two Datalog programs with the same distinguished nullary output predicate O that serves as input. As before, we annotate the relation names of both programs P_1 and P_2 with index 1 and 2, respectively, and denote the obtained programs by P_1^* and P_2^*.

We now construct program P over the schema:

$$\sigma(P_1^*) \cup \sigma(P_2^*) \cup \{\text{Adom}^{(1)}, O^{(2)}, E^{(2)}\},$$

by combining the rules from P_1^*, P_2^*, and those mentioned below. First we add rules $Adom(x_j) \leftarrow X(x_1, \ldots, x_\alpha)$ for every relation $X^{(\alpha)} \in \sigma(P_1^*) \cup \sigma(P_2^*) \cup \{E^{(2)}\}$ and $j \in [\alpha]$. Further, we add:

$$O(x, y) \leftarrow O_2(), Adom(x), Adom(y). \tag{τ_1}$$

$$O(x, y) \leftarrow O_1(), E(x, y). \tag{τ_2}$$

$$O(x, y) \leftarrow O(x, w), E(w, y). \tag{τ_3}$$

Notice that new relation E is an extensional database relation, while O and Adom are intensional database relations. Next, we define a GHP $\mathcal{H}$. For this take a single 1-dimensional cube of $p = 2$ servers, say *cube 1*, and define $\chi(\tau) = 1$ for all rules in P. For rules in P_1^* and P_2^*, as well as the Adom producing rules, we define $\rho^\tau(1) = \emptyset$. For rule $^\tau 1$ we again define $\rho^{\tau_1}(1) = \emptyset$, for $^\tau 2$ and $^\tau 3$ we define $\rho^{\tau_2}(1) = \rho^{\tau_3}(1) = \{y\}$.

We claim that P is 2-bounded if, and only if, $P_1 \subseteq P_2$. Otherwise, a GHP exists in $\mathcal{H}$ for which the number of rounds depends on the size of the input, particularly on the size of relation E.

(If) Let $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ be an arbitrary economic policy from $\mathcal{H}$. We observe that after a single round, programs P_1^* and P_2^*, as well as relation *Adom* are fully computed on both servers. (The latter is due to our choice $\rho^\tau(1) = \emptyset$ for the involved rules.) During the first round, O-facts may be produced by rule τ_2. After this first round, several facts will be communicated, particularly the $\boldsymbol{C}$-consumable facts derived with rules from P_1^* and P_2^*, as well as the facts with relation name Adom and O. Since these relations are computed on both servers, no server receives a new fact (particularly due to $P_1 \subseteq P_2$). Hence, the fixpoint is reached and no further communication steps are needed.

(Only if) Since $P_1 \not\subseteq P_2$, some instance I' exists, with $O() \in P_1(I')$ and $O() \notin P_2(I')$. We convert instance I' to an instance for P, by annotating the relations with respective index, and add a relation E, with the chain $E(0, 1), E(1, 2), \ldots, E(m - 1, m)$ for some integer m.

We define a specific GHP from $\mathcal{H}$. For this, let T^I be the transitive closure relation of E^I. As hash function, we choose $h_1^1(\{i\}) = i \bmod 2$. We note that, by the choice of h_1^1, $E(0, 1)$ is consumed at server 1, $E(1, 2)$ is consumed at server 2, $E(2, 3)$ is consumed again at server 1, etc. We observe that server 1 derives the fact $O(0, 1)$ in the first round and sends it to server 2. Then, server 2 can derive (in the second round) the fact $O(1, 2)$, based on $O(0, 1)$ and the fact $E(1, 2)$ which it had already received in an earlier round. Now, a straightforward inductive argument shows that server $i \bmod 2$ receives fact $O(i - 2, i - 1)$ for the first time in round i, and thus that we need $\Omega(m)$ rounds to reach a fixpoint. So for m large enough we need more than k rounds. □

Proposition 10 *k-BOUNDEDNESS(PureDatalog, $\mathcal{E}_{indep}$) is undecidable if $k \geq 2$.*

Proof The proof is again by a reduction from the undecidable Datalog containment problem. Given two Datalog programs P_1, P_2 over single output relation, which serve as input for Datalog containment. We construct program P by taking all rules in P_1, where all IDB relations are annotated by index 1, and all rules in P_2, where IDB relations marked with index 2. Here we assume that O_1 is the output predicate for P_1, and O_2 for P_2. We add the following rules, with fresh relation names $\{D_i \mid i \in [k]\}$:

$$D_1() \leftarrow O_1().$$
$$D_k() \leftarrow O_2().$$

And for each $i \in \{1, \ldots, k-1\}$ we add the rule:

$$D_{i+1}() \leftarrow D_i().$$

We take an economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ over a two-node network. The consumption and production policies are defined as follows:

- All relations with index 1 are consumed and produced by server 1;
- All relations with index 2 are consumed and produced by server 2;
- All relations D_i with even i are produced at server 2 and consumed at server 1;
- All relations D_i with odd i are produced at server 1 and consumed at server 2; and
- Relation D_k is produced at server 2 (even if k is odd).

Next, we show that $\boldsymbol{E}$ is k-bounded if and only if $P_1 \subseteq P_2$.

(Only if) Suppose $P_1 \not\subseteq P_2$. Then let I be an instance with $O_1() \in P_1(I)$ and $O_2() \notin P_2(I)$. In the first round, server 1 derives fact $D_1()$, which needs to be communicated in the next round (round 2) to server 2. In round 2, server 2 receives D_1 and produces D_2, which needs to be communicated in the next round (round 3) to server 1. Since server 1 and server 2 cannot produce facts for relations D_i in another way than via rule $D_{i+1}() \leftarrow D_i()$, they are deemed to continue this exchange of facts till D_k is produced (at round k) and received by its consuming server (at round $k+1$). Policy $\boldsymbol{E}$ is thus clearly not k-bounded.

(If) Suppose $P_1 \subseteq P_2$. On every instance I, server 1 computes $P_1(I)$, and server 2 computes $P_2(I)$. We distinguish between three cases: If $P_2(I)$ is empty, then the network fixpoint is reached immediately after the first round. If $P_1(I)$ is empty and P_2 is not, then the fact $D_k()$ is derived at server 2 and *may* have to be send to server 1 in the round (if k is even). Since D_1 is not derived, and will not be derived after receiving $D_k()$, the network fixpoint is reached after at most two rounds. The more interesting case is when $P_1(I)$ is not empty. Then server 1 derives fact $D_1()$, which triggers the consecutive exchange of $D_i()$ facts between the two servers as deribed in the only-if case of the proof, except that, when receiving fact $D_{k-1}()$ (in round k), the fact $D_k()$ is already known by its consuming server (i.e., server 1 if k is even, server 2 otherwise). Therefore, the network fixpoint is reached already in round k, which concludes the proof. □

Result (4) follows from the syntactical characterization shown in the next subsection. Towards this characterization, we first give a general characterization of 1-boundedness for strongly supporting policies.

Let P be a Datalog program and $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ an economic policy. We denote by $\boldsymbol{P}^*$ the policy obtained by removing from every $\boldsymbol{P}(\boldsymbol{f})$ any server s for which no rule instantiation $v(\tau)$ exists with $v(head_\tau) = \boldsymbol{f}$, $v(body_\tau) \subseteq \mathsf{facts}_{\boldsymbol{C}}(s)$, with $v(body_\tau)$ being all P-derivable. Intuitively, $\boldsymbol{P}^*(\boldsymbol{f})$ removes those servers that are allowed to produce $\boldsymbol{f}$, but cannot due to limitations of the consumption policy $\boldsymbol{C}$. Notice that if $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ is strongly supporting for P, then so is $\boldsymbol{E} = (\boldsymbol{P}^*, \boldsymbol{C})$, since we have not removed the support of any rule instantiation.

Proposition 11 *Let P be a Datalog program and $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ a strongly supporting economic policy for P. $\boldsymbol{E}$ is 1-bounded if and only if for every P-derivable intensional database fact $\boldsymbol{f}$: (1) $|\boldsymbol{C}(\boldsymbol{f})| \leq 1$; and (2) $|\boldsymbol{C}(\boldsymbol{f})| = 1$ implies $\boldsymbol{C}(\boldsymbol{f}) = \boldsymbol{P}^*(\boldsymbol{f})$.*

Proof (If) All intensional database facts derived during the distributed evaluation are P-derivable. Consider a rule instantiation (τ, v) that is fired on some server s and produces fact $\boldsymbol{f} = v(head_\tau)$. Then, condition (1) tells us that $|\boldsymbol{C}(\boldsymbol{f})| \leq 1$. If $|\boldsymbol{C}(\boldsymbol{f})| = 0$, then $\boldsymbol{f}$ is not consumed anywhere and thus will not be communicated. If $|\boldsymbol{C}(\boldsymbol{f})| = 1$, condition (2) tells us that $\boldsymbol{C}(\boldsymbol{f}) = \boldsymbol{P}^*(\boldsymbol{f})$. But since $s \in \boldsymbol{P}^*(\boldsymbol{f})$, this implies that $\boldsymbol{C}(\boldsymbol{f}) = \{s\}$. Hence, s is the only server that consumes $\boldsymbol{f}$, and $\boldsymbol{f}$ does not have to be sent to another server. Thus indeed $\boldsymbol{E}$ is 1-bounded. Notice that extensional database facts are never communicated after round 1.

(Only if) Suppose that $\boldsymbol{E}$ is 1-bounded. Let $\boldsymbol{f}$ be a $\boldsymbol{P}$-derivable fact. Since $\boldsymbol{E}$ is strongly-supporting, it is parallel-correct, thus $\boldsymbol{f}$ is derived at some server s over some instance I in round 1. In particular, $s \in \boldsymbol{P}^*(\boldsymbol{f})$. If $\boldsymbol{C}(\boldsymbol{f}) \not\subseteq \{s\}$, then $\boldsymbol{f}$ needs to be communicated by s, which enforces another round and contradicts 1-boundedness. Hence, $\boldsymbol{C}(\boldsymbol{f}) \subseteq \{s\}$ and $|\boldsymbol{C}(\boldsymbol{f})| \leq 1$. Assume $\boldsymbol{C}(\boldsymbol{f}) = \{s\}$ and suppose that there exists some $s' \in \boldsymbol{P}^*(\boldsymbol{f}) \setminus \{s\}$. Then, over some instance J, $\boldsymbol{f}$ is derived in s' in round 1, and then needs to be communicated to s, which again contradicts 1-boundedness. □

7.1 Weakly Pivoting GHPs

We present a necessary and sufficient syntactic condition for 1-boundedness of GHP families. Here, for atom A and set of variables $X \subseteq vars(A)$, we denote by $pos_A(X)$ the set of positions in A containing variables from X.

Definition 6 (**Pivoting Relation**) A relation R is *pivoting for GHP family $\mathcal{H}$* if for every two atoms A_1, A_2 (in rules τ_1, τ_2 respectively) over R, and for all dimensions i of cube $\chi(\tau_1)$ with $p_{\chi(\tau_1),i} > 1$:

1. $\emptyset \subsetneq \rho^{\tau_1}(i) \subseteq vars(A_1)$;
2. $\chi(\tau_1) = \chi(\tau_2)$; and
3. $pos_{A_1}(\rho^{\tau_1}(i)) = pos_{A_2}(\rho^{\tau_2}(i))$.

Intuitively, if R is pivoting, then every rule that sends R tuples will send each R tuple to exactly one server, and the rules agree on this server.

Example 7 For example, take the program

$$\tau_1 : \ T(x, y) \leftarrow R(x, y).$$
$$\tau_2 : \ T(x, y) \leftarrow T(x, z), R(z, y).$$
$$\tau_3 : \ O(y) \leftarrow T(x, y), S(x).$$

and the GHP over the single one-dimensional cube (*cube* 1). We define $\chi(\tau_1) = \chi(\tau_2) = \chi(\tau_3) = 1$ and $\rho^{\tau_1}(1) = \rho^{\tau_2}(1) = \rho^{\tau_3}(1) = \{x\}$. Let map^1 be the identity mapping. Here, S and T are pivoting relations; O and R are not pivoting.

Definition 7 (**Pivoting/Weakly pivoting**) We say that a GHP family is *pivoting* (*weakly pivoting*, resp.) for P if all (all P-consumable,[5] resp.) intensional database relations are pivoting.

The program from Example 7 is weakly pivoting. For pure programs we can test whether a GHP family is weakly pivoting in polynomial time, since we need to go over all P-consumable intensional database relations (for pure programs, these are all relations occurring in the body of a rule), and then for each such relation R test all pairs of atoms over R. This observation, along with the proposition below—that shows that weakly pivoting is a necessary and sufficient condition for 1-boundedness—implies that deciding 1-boundedness for GHP families is indeed in PTIME.

Proposition 12 *Let P be a pure Datalog program, and $\mathcal{H}$ a GHP family. Then, $\mathcal{H}$ is 1-bounded for P if and only if it is weakly pivoting for P.*

Proof (If) Let $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ be an arbitrary economic policy in $\mathcal{H}$ that is weakly pivoting for P. We show that $\mathcal{H}$ is 1-bounded for P by making use of Proposition 11.

For this, let $\boldsymbol{f}$ be an arbitrary P-derivable fact. We first deal with the case when $\boldsymbol{f}$ is not P-consumable. Due to pureness of P, the latter implies that no rule in P exists with a body atom that can match with $\boldsymbol{f}$. It follows from the definition of GHP that $\boldsymbol{f}$ is not $\boldsymbol{C}$-consumable (i.e., $|\boldsymbol{C}(\boldsymbol{f})| = 0$) and thus that the conditions in Proposition 11 are true for $\boldsymbol{f}$.

For the remainder of this direction of the proof we have to deal only with the case when $\boldsymbol{f}$ is P-derivable and P-consumable, which implies $|\boldsymbol{P}(\boldsymbol{f})| \geq 1$, and $|\boldsymbol{C}(\boldsymbol{f})| \geq 1$. Recall that a server s is in $\boldsymbol{C}(\boldsymbol{f})$ iff there is a rule $\tau \in P$, atom $A \in body_\tau$, and valuation v, with $v(A) = \boldsymbol{f}$. Analogously, server s is in $\boldsymbol{P}(\boldsymbol{f})$ iff there is a rule $\tau \in P$ and valuation v, with $v(A) = \boldsymbol{f}$ for $A = head_\tau$. Moreover, per definition 6 for weakly pivoting programs, in both cases server s is uniquely determined (for given τ and A) by the parameters $\chi(\tau)$ and $pos_A(\rho^\tau(i))$, for all dimensions i of $\chi(\tau)$. (We ignore $\mathsf{map}^{\chi(\tau)}$, which is fixed for $\chi(\tau)$.)

[5]Recall that a relation R is P-consumable if for some instance I, during the evaluation of P over I, a rule is fired that requires a fact from R.

Next, we show that $|\boldsymbol{C}(\boldsymbol{f})| = 1$. For this, arbitrary elements $s_1, s_2 \in \boldsymbol{C}(\boldsymbol{f})$. Then, due to Definition 6 it follows directly that $s_1 = s_2$. Hence, indeed $|\boldsymbol{C}(\boldsymbol{f})| = 1$, which corresponds to condition (1) of Proposition 11. The argument for condition (2) of Proposition 11, that $s_1 = s_2$ for every pair of servers $s_1 \in \boldsymbol{P}(\boldsymbol{f})$ and $s_2 \in \boldsymbol{C}(\boldsymbol{f})$ (which implies $\boldsymbol{P}(\boldsymbol{f}) = \boldsymbol{C}(\boldsymbol{f})$) is analogous. We thus conclude from Proposition 11 that $\boldsymbol{E}$ is 1-bounded. Then, from the generality of the argument it follows that $\mathcal{H}$ is 1-bounded.

(Only If) Let R be an arbitrary consumable intensional predicate name in P. We argue that R is pivoting by showing conditions (1), (2), and (3) from Definition 6 by contraposition. Notice that all facts over R are both P-consumable and P-derivable, due to pureness of P and because R is intensional.

First suppose that (1) fails for some τ_1, i, and atom A_1 over R. If A_1 is a body atom it follows immediately that all facts $\boldsymbol{f}$ over R are replicated in the construction for $\boldsymbol{C}$ over dimension i, which implies $|\boldsymbol{C}(\boldsymbol{f})| \geq 2$ and thus contradicts with 1-boundedness due to Proposition 11. Now assume A_1 is the head of τ_1. If $\rho^{\tau_1}(i) = \emptyset$ it follows that all rule instantiations for τ_1 are replicated over dimension i, and thus that $\boldsymbol{P}^*(\boldsymbol{f}) > 1$ for all facts $\boldsymbol{f}$ matching the head of τ_1. Since such a fact $\boldsymbol{f}$ is $\boldsymbol{C}$-consumable and $p_i > 1$ (which implies $|\boldsymbol{C}(\boldsymbol{f})| \geq 2$, this again contradicts 1-boundedness. For the case where $\rho^{\tau_1}(i) \neq \emptyset$, a similar argument holds: Take $x \in \rho^{\tau_1}(i) \setminus \mathit{vars}(A_1)$ and consider two valuations mapping all variables on the same value, except for x. We can now chose the hash functions for ρ^{τ_1} so that both rule instantiations are fired on different servers (due to $p_1 > 1$), and thus again $|\boldsymbol{P}^*(\boldsymbol{f})| > 1$, for some $\boldsymbol{C}$-consumable fact $\boldsymbol{f}$, which contradicts 1-boundedness.

For condition (2), $\chi(\tau_1) \neq \chi(\tau_2)$ allows choosing valuations for τ_1 and τ_2 that agree on the A_1 and A_2 (due to pureness), and then hash functions can be chosen so that both are fired on different servers. Since all matching facts are $\boldsymbol{C}$-consumable, this would contradict 1-boundedness (since $|\boldsymbol{P}(\boldsymbol{f})| > 1$ implies $\boldsymbol{P}(\boldsymbol{f}) \neq \boldsymbol{C}(\boldsymbol{f})$).

Finally, for condition (3) we can assume $\chi(\tau_1) = \chi(\tau_2)$ by case distinction. Then, for $pos_{A_1}(\rho^{\tau_1}(i)) \neq pos_{A_2}(\rho^{\tau_2}(i))$ the observation follows analogously. □

We remark that Proposition 12 cannot be easily generalized. For example, one cannot replace GHP families by strongly supporting policies, since then facts $\boldsymbol{f}$ that are not P-consumable may still be $\boldsymbol{C}$-consumable (i.e., $\boldsymbol{C}(\boldsymbol{f}) \neq \emptyset$). Reasoning about the latter requires a concrete representation mechanism for policies. (See also [7] for a discussion on this matter.) Further, it is unclear what the complexity becomes for testing 1-boundedness under general (not necessarily pure) Datalog, since then it is required to reason about P-derivability of facts.

Example 8 For an example showing that not every 1-bounded GHP is weakly pivoting, consider the following *non-pure* Datalog program P:

$$R(x, x) \leftarrow S(x, x). \qquad T(x, y) \leftarrow R(x, y). \qquad T(x, y) \leftarrow T(z, x), R(z, y).$$

and GHP family $\mathcal{H}$ over a single one-dimensional cube 1. Let map^1 be the identity mapping, $\chi(\tau) = 1$ and $\rho^{\tau}(1) = \{x\}$ for all rules τ. Clearly, $\mathcal{H}$ is not weakly pivoting. Nevertheless, it can be shown that $\mathcal{H}$ is 1-bounded, which follows from the observation that only single-valued rule instantiations can satisfy under P.

7.2 Weakly Pivoting Datalog

We have so far looked at whether a given GHP family is 1-bounded. In this section, we ask: *which Datalog programs admit a 1-bounded policy?*

If $A = R(\boldsymbol{x})$ is an atom, we use $A[i]$ to denote the variable/constant in atom A in position i. We naturally extend $A[\cdot]$ to map tuples of positions (that take values from the set $\{1, \ldots, ar(R)\}$) into tuples of variables/constants. For example, if $A = R(x_1, x_2, x_3)$ and $\boldsymbol{b} = (1, 3)$, then $A[\boldsymbol{b}] = (A[1], A[3]) = (x_1, x_3)$.

Definition 8 (**Pivot Base**) Let P be a Datalog program, and let $\sigma \subseteq \text{IDB}(P)$. Let $\boldsymbol{\beta}$ be a function that takes as input some relation name $R \in \sigma$ and outputs a non-empty tuple with values in $[ar(R)]$. We say that $\boldsymbol{\beta}$ is a *pivot base* for σ if:

– For every rule $\tau \in P$ and for every pair of atoms $R(\boldsymbol{x})$, $S(\boldsymbol{y})$ in $\{head_\tau\} \cup body_\tau$, such that $R, S \in \sigma$, we have $R(\boldsymbol{x})[\boldsymbol{\beta}(R)] = S(\boldsymbol{y})[\boldsymbol{\beta}(S)]$.

A Datalog program P is *pivoting* (*weakly pivoting*, resp.) if it has a pivot base for all relations in $\text{IDB}(P)$ (for all relations in $\text{IDB}(P)$ that occur in the body of some rule in P).

Example 9 Consider the left-linear TC example, and let $\sigma = \{T\}$. Suppose we choose $\boldsymbol{\beta}(T) = (1)$. Then $\boldsymbol{\beta}$ is a pivot base for σ, since for the recursive rule and the only pair of T-atoms $T(x, y)$, $T(x, z)$ we have $T(x, y)[\boldsymbol{\beta}(T)] = T(x, y)[1] = (x)$, and $T(x, z)[\boldsymbol{\beta}(T)] = T(x, z)[1] = (x)$. Since T is the only intensional database relation, left-linear TC is pivoting.

Next, consider the left-linear TC with an extra rule:

$$T(x, y) \leftarrow R(x, y). \qquad T(x, y) \leftarrow T(x, z), R(z, y). \qquad U(y) \leftarrow T(x, y).$$

Here, there are two intensional database relations, but only T occurs in the body of a rule. The pivot base $\boldsymbol{\beta}$ from before is still a pivot base for $\{T\}$; hence the program is weakly pivoting. However, there is no pivot base for to $\{T, U\}$, which means that the program is not pivoting.

The concept of pivoting Datalog was first introduced for single rule programs [35] and then generalized to full Datalog [28] where it is called *generalized pivoting*. The definition in [28] is based on a rather complex argument over fractional weight-mappings, but relates to pivoting in that every generalized pivoting Datalog program is pivoting for *all* intensional database relations. For pure Datalog these notions are equivalent. The proposition below shows that for pure Datalog, a weakly pivoting program admits a weakly pivoting (and thus 1-bounded) GHP family.

Proposition 13 *Let P be a pure Datalog program and $p \geq 2$. There is a 1-bounded GHP family for P if and only if P is weakly pivoting.*

In the below propositions, we show slightly stronger results. We start with the if-direction of Proposition 13, which follows from Proposition 12 and the below Proposition 14. Henceforth, we use for an atom A with relation symbol R, and tuple

$\boldsymbol{b}$ of integers from $[ar(R)]$, the notation $vars_A[\boldsymbol{b}]$ to denote the set of variables in A on the positions defined by $\boldsymbol{b}$.

Proposition 14 *Let P be a pure and weakly pivoting Datalog program. For every p there is a weakly pivoting GHP family for P over $[p]$.*

Proof Take a weak pivot base $\boldsymbol{\beta}$ for P. We construct GHP $\boldsymbol{E}$ over network $[p]$ by considering a single cube, *cube 1*, with only one dimension. We choose $\chi(\tau) = 1$ for every $\tau \in P$, and map^1 as the mapping from the single-point coordinates to servers in $[p]$ that expresses identity. Now for rules $\tau \in P$ having no atom with associated pivot base, we define $\rho^\tau(1) = \emptyset$; for all other rules we define $\rho^\tau(1) = vars_A[\boldsymbol{\beta}(R)]$, with R the relation symbol of A. It is easy to see that $\mathcal{H}$ is indeed weakly pivoting. □

For the only-if direction, we introduce the concept of straggler. Let P be a Datalog program and $\boldsymbol{E}$ a strongly supporting economic policy for P over $[p]$. We call $s \in [p]$ a *straggler for an intensional relation R* if either $s \in \boldsymbol{C}(\boldsymbol{f})$ for all facts $\boldsymbol{f}$ of R or $s \in \boldsymbol{P}^*(\boldsymbol{f})$ for all facts $\boldsymbol{f}$ of R. A straggler thus is a server that consumes or produces an entire relation.

The only-if direction of Proposition 13 then follows from Proposition 15 and Proposition 16, which are given below.

Proposition 15 *Let P be a pure Datalog program. If $\mathcal{H}$ is a weakly pivoting GHP for P over a network where $p \geq 2$, then every $\boldsymbol{E} \in \mathcal{H}$ is without stragglers for P-consumable intensional database relations.*

Proof To show that $\boldsymbol{E}$ has no stragglers, we recall that for a weakly pivoting policy (a) $|\rho^\tau(i)| > 0$ for every rule τ and dimension of the cube $\chi(\tau)$, and that (b) for every atom A over a P-consumable intensional relation, $\rho^\tau(i) \subseteq vars(A)$. Condition (b) implies that the facts from P-consumable intensional relations are consumed and produced at just one server. Condition (a) implies that the hash functions used by $\boldsymbol{E}$ are surjective and thus that, for every intensional P-consumable relation R, for at least some pair of facts $\boldsymbol{f}$ and $\boldsymbol{g}$ over R we have $\boldsymbol{C}(\boldsymbol{f}) \neq \boldsymbol{C}(\boldsymbol{g})$, and analogously, for some pair we have $\boldsymbol{P}^*(\boldsymbol{f}) \neq \boldsymbol{P}^*(\boldsymbol{g})$. In other words, not all facts over R are consumed or produced on the same server, which proves the desired result. □

Proposition 16 *Let P be a pure and not weakly pivoting Datalog program. Then, every strongly supporting economic policy that is 1-bounded has a straggler for some consumable intensional database relation in P.*

In the remainder of this section, we prove Proposition 16. For this, we introduce the notion of policy key for economic policy $\boldsymbol{E} = (\boldsymbol{C}, \boldsymbol{P})$ and Datalog program P. Let R be some intensional database relation and $\boldsymbol{\gamma}$ a tuple of integers in $|ar(R)|$. Then $\boldsymbol{\gamma}$ is called a *policy key* for R in $\boldsymbol{E}$, if for all facts $\boldsymbol{f}, \boldsymbol{g}$ over R, $\boldsymbol{f}[\boldsymbol{\gamma}] = \boldsymbol{g}[\boldsymbol{\gamma}]$ implies $\boldsymbol{P}^*(\boldsymbol{f}) = \boldsymbol{P}^*(\boldsymbol{g}) = \{s\}$, for some server s; and $\boldsymbol{C}(\boldsymbol{f}) = \boldsymbol{C}(\boldsymbol{g}) = \emptyset$ or $\boldsymbol{C}(\boldsymbol{f}) = \boldsymbol{C}(\boldsymbol{g}) = \{s\}$ (with s denoting the same server as before). When $\boldsymbol{E}$ is clear

from the context we omit mentioning $\boldsymbol{E}$ and say that $\boldsymbol{\gamma}$ is a policy key for R. We call $\boldsymbol{\gamma}$ *empty* if $\boldsymbol{\gamma} = ()$.

For 1-bounded and strongly supporting economic policies, all $\boldsymbol{C}$-consumable intensional database relations have a policy key (namely the tuple containing all positions), which follows immediately from Proposition 11.

Lemma 5 *For pure Datalog program P, and 1-bounded strongly supporting economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ for P, are equivalent for each intensional predicate R:*

1. *R has empty policy key in $\boldsymbol{E}$;*
2. *$\boldsymbol{E}$ has a straggler for R.*

Proof For (1) $\Rightarrow$ (2): By definition of policy key, there is a server s that belongs to $\boldsymbol{P}^*(\boldsymbol{f})$ for every fact $\boldsymbol{f}$ with predicate R. Hence, s is a straggler for R.

For (2) $\Rightarrow$ (1): Let s be the assumed straggler. If $s \in \boldsymbol{C}(\boldsymbol{f})$ for every $\boldsymbol{f}$ over R, then 1-boundedness (particularly Proposition 11) implies $\{s\} = \boldsymbol{P}^*(\boldsymbol{f})$ and $\{s\} = \boldsymbol{C}(\boldsymbol{f})$. We conclude that R has empty policy key. If $s \in \boldsymbol{P}^*(\boldsymbol{f}) \subseteq \boldsymbol{P}(\boldsymbol{f})$ for all $\boldsymbol{f}$, then it follows from 1-boundedness that $\{s\} = \boldsymbol{P}(\boldsymbol{f})$ and that $\boldsymbol{C}(\boldsymbol{f})$ equals $\{s\}$ or is empty. It then follows directly that $\{s\} = \boldsymbol{P}^*(\boldsymbol{f})$. □

We can also show the following technical results regarding policy keys.

Lemma 6 *Let $\boldsymbol{E}$ be an economic policy and R a relation in the schema of $\boldsymbol{E}$. If $\boldsymbol{\gamma_1}$ and $\boldsymbol{\gamma_2}$ are policy keys for R, then every tuple $\boldsymbol{\gamma}$ having all integers that are in both $\boldsymbol{\gamma_1}$ and $\boldsymbol{\gamma_2}$, is a policy key for R.*

Proof Let $\boldsymbol{\gamma_1}, \boldsymbol{\gamma_2}, \boldsymbol{\gamma}$ be as in the proposition, and $\boldsymbol{f}, \boldsymbol{g}$ be arbitrary facts over relation R, with $\boldsymbol{f}[\boldsymbol{\gamma}] = \boldsymbol{g}[\boldsymbol{\gamma}]$.

We construct fact $\boldsymbol{f}'$ over R, by taking as values on positions mentioned in $\boldsymbol{\gamma_1}$ the same values as in $\boldsymbol{f}$, and for values on positions in $\boldsymbol{\gamma_2}$ the same values as in $\boldsymbol{g}$. Notice that the construction is well-defined, because $\boldsymbol{\gamma}$ contains *all* positions from the intersection.

Now it is easy to see that $\boldsymbol{\gamma}$ is indeed a key. Specifically because $\boldsymbol{C}(\boldsymbol{f}) = \boldsymbol{C}(\boldsymbol{f}') = \boldsymbol{C}(\boldsymbol{g})$ and $\boldsymbol{P}^*(\boldsymbol{f}) = \boldsymbol{P}^*(\boldsymbol{f}') = \boldsymbol{P}^*(\boldsymbol{g})$. □

Lemma 7 *Let P be a pure Datalog program and $\boldsymbol{E}$ a strongly supporting economic policy. Let $A_1 = R_1(\boldsymbol{x_1})$ and $A_2 = R_2(\boldsymbol{x_2})$ be two atoms occurring in some rule $\tau \in P$, with R_1 and R_2 be intensional predicates. Let $\boldsymbol{\gamma_1}$ and $\boldsymbol{\gamma_2}$ be minimal keys for $R1$ and $R2$, respectively. Then, $vars_{A_1}[\boldsymbol{\gamma_1}] = vars_{A_2}[\boldsymbol{\gamma_2}]$.*

Proof Let $\boldsymbol{E}$ be the economic policy from the lemma. Let $\boldsymbol{\gamma}$ be a tuple consisting of all positions in $pos_{A_1}(vars_{A_1}[\boldsymbol{\gamma_1}] \cap vars_{A_2}[\boldsymbol{\gamma_2}])$ ordered arbitrarily. We shall show that $\boldsymbol{\gamma}$ is a key for R_1 (and due to symmetry for R_2). Then, the desired property

$vars_{A_1}[\boldsymbol{\gamma_1}] = vars_{A_2}[\boldsymbol{\gamma_2}]$ follows from Lemma 6 and the assumption that $\boldsymbol{\gamma_1}$ and $\boldsymbol{\gamma_2}$ are minimal.

To show that $\boldsymbol{\gamma}$ is indeed a key for R_1, let $\boldsymbol{f}$ and $\boldsymbol{g}$ be two arbitrarily chosen facts over R_1, with $\boldsymbol{f}[\boldsymbol{\gamma}] = \boldsymbol{g}[\boldsymbol{\gamma}]$. Let v_1 be a valuation for τ, with $v_1(A_1) = \boldsymbol{f}$; and let v_2 be a valuation for τ with $v_2(A_1) = \boldsymbol{g}$. We consider also a valuation v for rule τ, with $v(x) = v_1(x)$ for all $x \in vars(A_1) \setminus vars_{A_2}[\boldsymbol{\gamma_2}]$ and $v(x) = v_2(x)$ for all other variables. (Recall that all these valuations exist because P is pure.)

Since R_1 and R_2 have a key, every fact $\boldsymbol{h}$ with relation name R_1 or R_2 is associated to a unique server s, with $\boldsymbol{P}^*(\boldsymbol{h}) = \{s\}$, and $\boldsymbol{C}(\boldsymbol{h}) = \emptyset$ (if it is not consumed) or $\boldsymbol{C}(\boldsymbol{f}) = \{s\}$ (if it is consumed). Therefore, and since $\boldsymbol{E}$ is strongly supporting, the server associated to $v_2(A_1)$ is the same server as associated to $v_2(A_2)$. Let's call this server s_2. For the same reason, the server associated to $v(A_1)$ is the same server as associated to $v(A_2)$. Let's call this server s_3. Now since $v(A_2)$ and $v_2(A_2)$ agree on their key (that is, $v(vars_{A_2}[\boldsymbol{\gamma_2}]) = v_2(vars_{A_2}[\boldsymbol{\gamma}2])$), it follows that $s_2 = s_3$.

To conclude the proof, we observe that $v_1(A_1)$ agrees with $v(A_1)$ on its key (particularly because v_1 and v_2 agree on the values for variables in $vars_{A_1}[\boldsymbol{\gamma_1}] \cap vars_{A_2}[\boldsymbol{\gamma_2}]$), and thus that the server s_2 is associated also to $v_1(A_1)$. In other words, $\boldsymbol{P}^*(\boldsymbol{f}) = \boldsymbol{P}^*(\boldsymbol{g})$ and $\boldsymbol{C}(\boldsymbol{f}) = \boldsymbol{C}(\boldsymbol{g})$, which proves that $\boldsymbol{\gamma}$ is indeed a key for R_1. □

Proof of Proposition 16 Suppose $\boldsymbol{E}$ is a strongly supporting economic policy that is 1-bounded. For the sake of contradiction, assume that $\boldsymbol{E}$ has no stragglers for any $\boldsymbol{C}$-consumable intensional database relations. Then, by Lemma 5 all P-consumable relations have only non-empty policy keys, which implies existence of minimal non-empty policy keys for the $\boldsymbol{C}$-consumable intensional relations. In turn, due to Lemma 7 we can take these minimal keys as pivot base. However, this contradicts the fact that P is not weakly pivoting. □

7.3 Bounded and Disjoint Evaluation

Sometimes we want to guarantee that, at the end of a computation, no two copies of the same fact have been derived at different servers. We call this property disjointness.

Definition 9 (**Disjointness**) Let P be a Datalog program, and R an intensional relation name of P. We call an economic policy $\boldsymbol{E}$ for P *R-disjoint* if for every instance, every fact of R is produced in at most one server.

We study economic policies that are both 1-bounded *and* disjoint.

Proposition 17 *Let $P \in$ PureDatalog and $\mathcal{H}$ a GHP family for P. Then, $\mathcal{H}$ is 1-bounded, disjoint for P, and without stragglers for intensional database relations, if and only if, $\mathcal{H}$ is pivoting.*

For the proof, we use the next auxiliary result.

Lemma 8 *An economic policy $\boldsymbol{E} = (\boldsymbol{C}, \boldsymbol{P})$ that is 1-bounded, disjoint, strongly supporting and without stragglers for intensional database relations of the associated Datalog program has non-empty minimal keys for these relations.*

Proof Due to 1-boundedness and the absence of stragglers intensional database relations, it follows from Lemma 5 that only non-empty keys (and thus non-empty minimal keys) for $\boldsymbol{C}$-consumable intensional database relations exist. □

We can now show the proof for Proposition 17.

Proof for Proposition 17 (If.) Since a pivoting GHP is also weakly pivoting, it follows from Proposition 12 and Proposition 15 that $\boldsymbol{E}$ is 1-bounded and without stragglers for P-consumable intensional relations.

For the remainder of the proof we observe that there exists $s \in \boldsymbol{P}^*(\boldsymbol{f})$ iff there is a rule $\tau \in P$ and valuation v such that $v(head_\tau) = \boldsymbol{f}$. Due to Definition 6 for pivoting GHPs, s is identified uniquely per rule τ by the combination $\chi(\tau)$ and $pos_{head_\tau}(\rho^\tau(i))$ for all i with $p_i \geq 1$.

For $s_1, s_2 \in \boldsymbol{P}^*(\boldsymbol{f})$, it follows again from Definition 6 for pivoting GHPs, that $s_1 = s_2$, thus $|\boldsymbol{P}^*(\boldsymbol{f})| = 1$. Hence, $\boldsymbol{E}$ is indeed disjoint. Due to surjectivity of the considered hash functions, it follows that $\boldsymbol{E}$ has no stragglers for intensional database relations.

(Only if.) 1-boundedness implies that $\mathcal{H}$ is weakly pivoting due to Lemma 8. It remains to show that condition (1) and (2) also hold for relations that are not consumable. The proof is again by contraposition and completely analogous to the proof of Proposition 12. Only now A_1 and A_2 must be head atoms, and we use disjointness to argue $|\boldsymbol{P}^*(\boldsymbol{f})| = 1$ for all facts $\boldsymbol{f}$ over R_1. □

Next, we show which programs admit a 1-bounded, disjoint policy.

Proposition 18 *Let $P \in$ PureDatalog. Then P is pivoting if, and only if, P admits a 1-bounded, strongly supporting, disjoint economic policy without stragglers for intensional database relations.*

Proof (If) The result follows immediately by Lemma 8 and Lemma7.

(Only If) The proof is analogous to the proof of Proposition 14 and takes the pivot base B for P to obtain a pivoting GHP for P. The result then follows from Proposition 17, which shows that this pivoting GHP is 1-bounded, strongly supporting, disjoint, and without stragglers for intensional database relations. □

Remark 1 The reader may wonder how the above concepts relate to the class of decomposable programs [36, 37]. A *decomposable* program is a (single rule) Datalog program that admits an evaluation strategy (via predicate restrictions) that is parallel-correct, 1-bounded, disjoint, and non-trivial. (Here non-triviality means that all servers do part of the work.) We did not consider the non-triviality property, but

instead require the absence of stragglers. Nevertheless, for GHPs, non-triviality is implied—at least for pure Datalog—by the use of surjective hash functions).

8 Conclusion

We have introduced a theoretical framework to reason about multi-round Datalog evaluation in a distributed setting. In this framework we study three properties: parallel-correctness, boundedness, and disjointness. There are many interesting questions left open. For example, it would be interesting to come up with restrictions on Datalog programs and economic policies, for which the mentioned properties are decidable. In fact, recent work by Neven et al. [27] extends our work in that direction. Among other results, they show that parallel-correctness is already undecidable even for heavily restricted fragments of Datalog, including monadic Datalog (for which the containment problem is decidable).

Another interesting direction for future work would be to study the problem of finding economic distribution policies with desired properties, which should not necessarily be harder than deciding properties over given policies. A related question then is which properties, besides the ones studied in the paper, are relevant in a practical context. One interesting option would be to define a fairness condition for economic policies, e.g., an instance independent notion of load-balancing; another option is to study bounds on the amount of communication needed to evaluate Datalog programs. Yet another direction is to consider smarter algorithms for local Datalog evaluation than semi-naive, by, for example, allowing to express unique-decomposition conditions (c.f., [5]) in the economic policy.

References

1. Foundations of databases: The logical level. In: Abiteboul, S., Hull, R., Vianu, V. (eds.) 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1995)
2. Afrati, F.N., Borkar, V.R., Carey, M.J., Polyzotis, N., Ullman, J.D.: Map-reduce extensions and recursive queries. In: EDBT '11, pp. 1–8 (2011)
3. Afrati, F.N., Papadimitriou, C.H.: The parallel complexity of simple chain queries. In: PODS '87, pp. 210–213 (1987)
4. Afrati, F.N., Ullman, J.D.: Optimizing joins in a map-reduce environment. In: EDBT '10, pp. 99–110 (2010)
5. Afrati, F.N., Ullman, J.D.: Transitive closure and recursive datalog implemented on clusters. In: EDBT '12, pp. 132–143 (2012)
6. Ameloot, T.J., Geck, G., Ketsman, B., Neven, F., Schwentick, T.: Data partitioning for single-round multi-join evaluation in massively parallel systems. SIGMOD Record **45**(1), 33–40 (2016)
7. Ameloot, T.J., Geck, G., Ketsman, B., Neven, F., Schwentick, T.: Parallel-correctness and transferability for conjunctive queries. J. ACM **64**(5), 36:1–36:38 (2017)
8. Ameloot, T.J., Ketsman, B., Neven, F., Zinn, D.: Datalog queries distributing over components. ACM Trans. Computa. Logic **18**(1), 5:1–5:35 (2017)
9. Beame, P., Koutris, P., Suciu, D.: Communication steps for parallel query processing. In: PODS '13, pp. 273–284 (2013)
10. Beame, P., Koutris, P., Suciu, D.: Skew in parallel query processing. In: PODS '14, pp. 212–223 (2014)
11. Chu, S., Balazinska, M., Suciu, D.: From theory to practice: Efficient join query evaluation in a parallel database system. In: SIGMOD '15, pp. 63–78 (2015)

12. Cosmadakis, S.S., Kanellakis, P.C.: Parallel evaluation of recursive rule queries. In: PODS '86, pp. 280–293. ACM, New York (1986)
13. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: OSDI '04, pp. 137–150 (2004)
14. Dewan, H.M., Stolfo, S.J., Hernández, M.A., Hwang, J.-J.: Predictive dynamic load balancing of parallel and distributed rule and query processing. In: SIGMOD '94, pp. 277–288 (1994)
15. Ganguly, S., Silberschatz, A., Tsur, S.: Parallel bottom-up processing of datalog queries. J. Logic Program. **14**(1&2), 101–126 (1992)
16. Ganguly, S., Silberschatz, A., Tsur, S.: A framework for the parallel processing of datalog queries. In: SIGMOD '90, pp. 143–152 (1990)
17. Geck, G., Ketsman, B., Neven, F., Schwentick, T.: Parallel-correctness and containment for conjunctive queries with union and negation. In: ICDT 2016, pp. 9:1–9:17 (2016)
18. Hadoop. http://hadoop.apache.org/
19. Halperin, D., de Almeida, V.T., Choo, L.L., Chu, S., Koutris, P., Moritz, D., Ortiz, J., Ruamviboonsuk, V., Wang, J., Whitaker, A., Xu, S., Balazinska, M., Howe, B., Suciu, D.: Demonstration of the myria big data management service. In: SIGMOD '14, pp. 881–884 (2014)
20. Kanellakis, P.C.: Logic Programming and Parallel Complexity, pp. 1–30. Springer, Berlin (1986)
21. Ketsman, B., Albarghouthi, A., Koutris, P.: Distribution policies for datalog. In: ICDT '18, pp. 17:1–17:22 (2018)
22. Ketsman, B., Neven, F.: Optimal broadcasting strategies for conjunctive queries over distributed data. Theory Comput. Syst. **61**(1), 233–260 (2017)
23. Ketsman, B., Suciu, D.: A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In: PODS '17, pp. 417–428 (2017)
24. Koutris, P., Beame, P., Suciu, D.: Worst-case optimal algorithms for parallel query processing. In: ICDT '16, pp. 8:1–8:18 (2016)
25. Koutris, P., Suciu, D.: Parallel evaluation of conjunctive queries. In: PODS '11, pp. 223–234 (2011)
26. Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, D.: Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In: AAAI '14, pp. 129–137 (2014)
27. Neven, F., Schwentick, T., Spinrath, C., Vandevoort, B.: Parallel-correctness and parallel-boundedness for datalog programs. In: ICDT'19, pp. 14:1–14:19 (2019)
28. Seib, J., Lausen, G.: Parallelizing datalog programs by generalized pivoting. In: PODS '91, pp. 241–251 (1991)
29. Seo, J., Park, J., Shin, J., Lam, M.S.: Distributed socialite: A datalog-based language for large-scale graph analysis. PVLDB **6**(14), 1906–1917 (2013)
30. Shaw, M., Koutris, P., Howe, B., Suciu, D.: Optimizing large-scale semi-naïve datalog evaluation in hadoop. In: Datalog 2.0, pp. 165–176 (2012)
31. Shkapsky, A., Yang, M., Interlandi, M., Chiu, H., Condie, T., Zaniolo, C.: Big data analytics with datalog queries on spark. In: SIGMOD '16, pp. 1135–1149 (2016)
32. Apache spark. http://spark.apache.org/
33. Ullman, J.D., Van Gelder, A.: Parallel complexity of logical query programs. Algorithmica **3**, 5–42 (1988)
34. Wang, J., Balazinska, M., Halperin, D.: Asynchronous and fault-tolerant recursive datalog evaluation in shared-nothing engines. PVLDB **8**(12), 1542–1553 (2015)
35. Wolfson, O.: Sharing the load of logic-program evaluation. In: DPDS '88, pp. 46–55 (1988)
36. Wolfson, O., Ozeri, A.: A new paradigm for parallel and distributed rule-processing. SIGMOD Rec. **19**(2), 133–142 (1990)
37. Wolfson, O., Silberschatz, A.: Distributed processing of logic programs. SIGMOD Rec. **17**(3), 329–336 (1988)
38. Xin, R.S., Rosen, J., Zaharia, M., Franklin, M.J., Shenker, S., Stoica, I.: Shark: SQL and rich analytics at scale. In: SIGMOD '13, pp. 13–24 (2013)
39. Zhang, W., Wang, K., Chau, S.-C.: Data partition and parallel evaluation of datalog programs. IEEE Trans. Knowl. Data Eng. **7**(1), 163–176 (1995)