Distribution Policies for Datalog

Bas Ketsman[1], Aws Albarghouthi[2], and Paraschos Koutris[2]

1 Hasselt University & transnational University of Limburg
2 University of Wisconsin-Madison

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.

1 Introduction

Modern data management systems—such as Spark [27, 33], Hadoop [16, 11], and others [17]—have extensively used parallelism to speed up query processing over massive volumes of data. Parallelism enables the distribution of computation into multiple machines, 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 [7, 8, 20, 19] have studied the tradeoff between synchronization (number of rounds) and communication cost, and have proposed and analyzed known and new parallel algorithms [2, 9]. Among these, the *Hypercube algorithm* [14, 2] 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] 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 machines. 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 for entire components of the input database that all facts must reside on the same server to ensure correctness of the computation.

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 machines 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 [30, 14, 31, 34]. There, Datalog evaluation strategies are based on the idea of partitioning the instantiations of the program rules among machines 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 [14, 2], and the decomposable strategies based on program restrictions [30].

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 machine 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. [6]; 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 in [32] in the context of decomposable programs.

2 Related Work

Parallel Complexity The parallel complexity of Datalog was first investigated in [10, 18]. Later work used the complexity class NC to theoretically capture which Datalog programs are efficiently parallelizable. Since Datalog evaluation is P-complete, it is unlikely that every Datalog program belongs in NC, which implies that certain Datalog programs may not be significantly sped up through parallelism. Ullman and Van Gelder [28] 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 [4] 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 tradeoff 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.

Decomposability The concept of predicate decomposability was first introduced by Wolfson and Silberschatz [32]. A predicate T is decomposable if there are $r > 1$ restricted copies $P_1, P_2, \dots, P_r$ 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 some input for 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 IDB 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 [30, 31] 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 IDB facts belong in the same partition block. Wolfson and Ozen [31] show that deciding whether a given Datalog program is decomposable is undecidable. Cohen and Wolfson [30] provide necessary and sufficient syntactic conditions for decomposability for sirups where the arity of the IDB 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. [34] present a more general framework that constructs partitionings of the rule instantiations.

Other Parallel Schemes In addition to decomposability, several frameworks for parallel recursive processing were introduced in the early 90s [30, 14, 31]. Wolfson [30] 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 [14, 13, 31], general schemes are introduced that parallelize the evaluation by partitioning the set of rule instantiations, and allowing for communication among the machines (decomposable and load sharing schemes need no communication). In [12], similar techniques are proposed 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.

Systems Recent work studies the implementation of Datalog (or fragments of Datalog) on modern shared-nothing distributed systems. Seo et al. [24] 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. [29] implement a variant of Datalog on the Myria system [17], focusing mostly on asynchronous evaluation and fault-tolerance. The BigDatalog system [26] 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 [3, 5, 25]. Motik et al. [22] 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 $R(a_1, \dots, a_k)$ over $U \subseteq \textbf{dom}$ is a tuple consisting of a relation name and a sequence of values from U. We say that $R(a_1, \dots, a_k)$ is *over* schema σ, if $R^{(k)} \in \sigma$. For a universe $U \subseteq \textbf{dom}$ and schema σ, we denote by $\mathsf{facts}(\sigma, U)$ the complete set of facts over σ and U. An *instance* I over σ and U is defined as a finite subset of $\mathsf{facts}(\sigma, U)$. 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 $[i]$.

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**. We say that $R(t_1, \ldots, t_k)$ is *over* schema σ, when $R \in \sigma$ and $k = ar(R)$.

A *Datalog rule* τ is of the form $(head_\tau, body_\tau)$, where $head_\tau$ is a single atom called the head of τ, and $body_\tau$ is a set of atoms called the body of τ. Henceforth, we shall also use the more conventional notation $R(\boldsymbol{x}) \leftarrow S_1(\boldsymbol{y_1}), \ldots, S_n(\boldsymbol{y_n})$, where $R(\boldsymbol{x})$ denotes the head, and all other atoms denote the body atoms of the rule. 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 $\text{EDB}(P) \subseteq \sigma$ we denote the relation names occurring only in the body of rules, and by $\text{IDB}(P) \subseteq \sigma$ all other relation names occurring in P. We further distinguish the names in $\text{IDB}(P)$ by calling some of them *output relations*, denoted $out(P) \subseteq \text{IDB}(P)$; all other IDB relations are *auxiliary.* We write $\sigma(P)$ to denote $\text{EDB}(P) \cup \text{IDB}(P)$.

Consider the directed graph where each node is an IDB predicate, 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).

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 $\text{EDB}(P)$. 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 $(v(head_\tau), v(body_\tau))$—sometimes abbreviated (τ, v)—as the instantiation of rule τ with valuation v. We say that a rule instantiation is *useless* if $v(head_\tau) \in v(body_\tau)$; otherwise, we say that it is *useful.*

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 semi-naive evaluation of P on some instance I a rule instantiation (τ, v) fires that requires $\boldsymbol{f}$. Both notions naturally generalize to atoms and predicate symbols, e.g., predicate symbol 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 $A \in body_\tau$.

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 $\mathsf{facts}(\sigma(P), \mathbf{dom})$, 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 $children_T(\boldsymbol{g}) = v(body_\tau)$. 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. W.l.o.g. we will consider only proof trees that do not use any useless rule instantiations.

We say that a proof tree T' is *entailed* by proof tree T for P, denoted $T' \sqsubseteq T$, if $\mathit{fringe}_{T'} \subseteq \mathit{fringe}_T$ and $\mathit{root}_{T'} = \mathit{root}_T$.

4 The Framework

Our framework considers a setting with p *servers* (or *machines*) 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 specify how the data (input and output facts) are distributed over the network. We borrow the definition of a distribution policy from [6]:

▶ **Definition 4.1** (Distribution Policy)**.** A *distribution policy* $\boldsymbol{P} = (U, \mathsf{facts}_{\boldsymbol{P}})$ over schema σ and network $[p]$ consists of a universe $U \subseteq \mathbf{dom}$ and a function $\mathsf{facts}_{\boldsymbol{P}} : [p] \to 2^{\mathsf{facts}(\sigma,U)}$ mapping servers to sets of facts over U and 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 [6], where the focus is on single-round query evaluation and policies that define only the initial data distribution over EDB facts, we consider a multi-round setting that allows the communication of intermediate facts.

▶ **Definition 4.2** (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 machines have the responsibility of producing a certain IDB fact. A consumption policy describes which machines need an EDB or IDB fact to satisfy the body of a rule instantiation. We sometimes make universe U explicit, by writing $(\boldsymbol{P}, \boldsymbol{C}; U)$ rather than $(\boldsymbol{P}, \boldsymbol{C})$.[1] We say that a fact $\boldsymbol{f}$ is $\boldsymbol{C}$*-consumable* if $\boldsymbol{C}(\boldsymbol{f}) \neq \emptyset$.

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 $\mathit{Policy}_R(\boldsymbol{x})$ to the body of τ. Here, predicate Policy_R refers to relation $\mathsf{facts}_{\boldsymbol{P}}(i)_{|\{R\}}$.

[1] Notice that mentioning U is redundant, but allows a slightly simpler notation, since $\boldsymbol{P}$ and $\boldsymbol{C}$ need not be specified explicitly to reference their universe U.

- Second, for every atom $S(\boldsymbol{y})$ in the body of τ, we add the predicate $Policy_S(\boldsymbol{y})$, where now $Policy_S$ refers to the relation $\mathsf{facts}_{\boldsymbol{C}}(i)_{|\{S\}}$.

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 "$\boldsymbol{t} \in \mathsf{facts}_{\boldsymbol{C}}(i)_{|\{S\}}$?". Of course, the complexity of such question depends on the expression mechanism of $\boldsymbol{P}$ and $\boldsymbol{C}$. We refer to [6] for a more through discussion of this matter.

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 $P_{\upharpoonright \boldsymbol{E}}(I, J)$ 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) [21]. 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 I_i the initial local instance of machine i. Let local_i^k be the instance on machine i right after the k-th communication phase.

We consider the following procedure: Initially, we set $\mathsf{local}_i^0 \leftarrow I_i$. Then, at the k-th round (for $k \geq 1$), we perform the following:

1. *Communication:* Every machine 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) $\boldsymbol{f} \in \mathsf{facts}_{\boldsymbol{C}}(j)$. Let rec_i^k be the facts received by machine i during the k-th communication phase.[2]
2. *Computation:* Every server computes the local fixpoint: if $k = 1$, then $\mathsf{fix}_i^k = P_{\upharpoonright \boldsymbol{E}}(\mathsf{rec}_i^k \cup \mathsf{local}_i^{k-1}, \mathsf{rec}_i^k \cup \mathsf{local}_i^{k-1})$, otherwise $\mathsf{fix}_i^k = P_{\upharpoonright \boldsymbol{E}}(\mathsf{rec}_i^k \cup \mathsf{local}_i^{k-1}, \mathsf{rec}_i^k \setminus \mathsf{local}_i^{k-1})$. Then, updates its local instance with $\mathsf{local}_i^k \leftarrow \mathsf{fix}_i^k$.

Intuitively, the algorithm terminates when, after a round is finished, for every server all locally derived facts that have to be send to some other server according to the consumption policy, were already send 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.

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 $\boldsymbol{f} \in \mathsf{facts}_{\boldsymbol{C}}(j)$ 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 5.3).

[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.

For 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 positive Datalog.

▶ **Example 4.3.** 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 EDB facts everywhere, and then produces/consumes each fact $T(a,b)$ at machine $h(a)$. 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 has $\boldsymbol{C}_2(R(a,b)) = \{h(a)\}$, $\boldsymbol{C}_2(T(a,b)) = \{h(b)\}$, and $\boldsymbol{P}_2(T(a,b)) = [p]$. This policy does not replicate the EDB facts, but it hash-partitions them according to the first attribute. Whenever a machine 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 machine will produce as many IDB 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 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 (positive) Datalog ensures monotonicity of economic policies for (positive) Datalog Programs.

▶ **Proposition 4.4.** *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.*

▶ Remark. While we study Datalog evaluation in the MPC model, where communication is synchronized by definition, it should be noted that all proposed evaluation strategies also work in settings where communication is done asynchronously. This follows naturally from the monotonicity of econonomy policies for (positive) Datalog programs. Also, most concepts we discuss remain relevant in the asynchronous setting, except of course for boundedness.

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 IDB relations. Henceforth, we are only interested in economic policies that generate the expected output.

▶ **Definition 5.1** (Parallel-correctness)**.** An economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C}; U)$ is *parallel-correct* for Datalog program P if, for every instance $I \subseteq \mathsf{facts}(\text{EDB}(P), U)$, $[P, \boldsymbol{E}](I) = P(I)|_{out(P)}$.

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 same relation symbol. We call this class of policies *value-independent*, denoted $\mathcal{E}_{indep}$, since the facts are mapped to machines only according to the relations they belong to. Value-independent policies allow a succinct representation by simply enumerating the IDB predicates of P and the subsets of $[p]$ where each relation is assigned.

We consider the following decision problem.

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

▶ **Theorem 5.2.** PC*(Datalog,* $\mathcal{E}_{indep}$*) is undecidable.*

In fact, we can show an even stronger result:

▶ **Lemma 5.3.** *Let P be an arbitrary Datalog program and $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C}; U)$ an economic policy over σ that is parallel-correct for P. Now let $\boldsymbol{f} \in \mathit{facts}(\sigma, U)$, and $\boldsymbol{C}'$ the consumption policy where $\boldsymbol{C}'(\boldsymbol{g}) = \boldsymbol{C}(\boldsymbol{g})$ for all $\boldsymbol{g} \in \mathit{facts}(\sigma, U) \setminus \{\boldsymbol{f}\}$ and $\boldsymbol{C}'(\boldsymbol{f}) \subsetneq \boldsymbol{C}(\boldsymbol{f})$. It is still undecidable whether $\boldsymbol{E}'$ is 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 5.4** (Support)**.** An instantiation of rule τ with valuation v is *supported* by economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ if there exists some machine $s \in [p]$ with $v(\mathit{head}_\tau) \in \mathsf{facts}_{\boldsymbol{P}}(s)$ and $v(\mathit{body}_\tau) \subseteq \mathsf{facts}_{\boldsymbol{C}}(s)$.

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 $v'(\mathit{head}_{\tau'}) = v(\mathit{head}_\tau)$ and $v'(\mathit{body}_{\tau'}) \subsetneq v(\mathit{body}_\tau)$.

$\mathcal{N}_P^{use}$: the set of all rule instantiations of P that are useful. Recall that an instantiation of rule τ with valuation v is useful if $v(\mathit{head}_\tau) \notin v(\mathit{body}_\tau)$.

$\mathcal{N}_P^{ess}$: 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 $\boldsymbol{g} = v(\mathit{head}_\tau)$ and $v(\mathit{body}_\tau) \subseteq \mathit{children}_T(\boldsymbol{g})$.

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

The following example demonstrates the different types of rule instantiations.

▶ **Example 5.6.** Let P be the left-linear transitive closure program from Example 4.3; 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:

$\boldsymbol{c = a}$**:** 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.

$\boldsymbol{c = b}$**:** in this case, the instantiation is useless, since $T(a, b)$ also belongs in the body. In some sense, this derivation is unnecessary, as we have already “discovered” the head fact.

$\boldsymbol{c \neq a, c \neq b}$**:** 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.

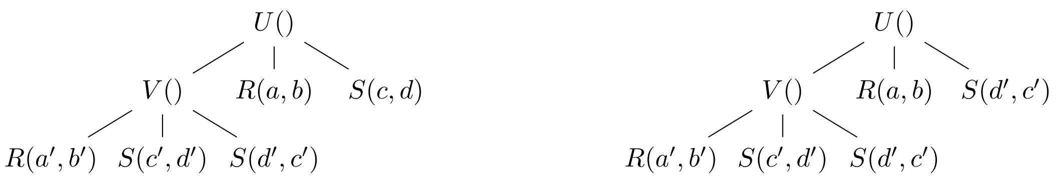

Figure 1 The two proof trees for the fact $\boldsymbol{f} = U(a, b)$.

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. We show in the Appendix that we can state the following two conditions for parallel-correctness:

▶ **Proposition 5.7.** *Let P be a Datalog program and $\boldsymbol{E}$ an economic policy. Then:*
1. *if $\boldsymbol{E}$ supports all minimal and useful rule instantiations in P, it is parallel-correct.*
2. *if $\boldsymbol{E}$ is parallel-correct for P, it supports all essential rule instantiations.*

▶ **Proposition 5.8.** *Let P be a Datalog program where each* IDB *predicate occurs only in the head of rules (i.e., P is a union of CQs). Then, $\mathcal{N}_P^{ess} = \mathcal{N}_P^{min} \cap \mathcal{N}_P^{use}$.*

Together with Proposition 5.7, the above proposition implies that a Datalog program where the body of each rule contains only EDB relations is parallel-correct if and only if it supports every minimal rule instantiation, or equivalently if and only 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 [6, 15]. We should emphasize here that [6, 15] 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 5.9.** 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 $\{V(), R(a, b), S(c, d)\} \subseteq \mathit{children}_T(U())$. 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 Figure 1. Because $c \neq d$, one of the facts $S(c', d'), S(d', c')$ must be different from $S(c, d)$ (In Figure 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 5.10.** This example shows that $\mathcal{N}_P^{ess} = \mathcal{N}_P^{min} \cap \mathcal{N}_P^{use}$ can hold for recursive programs. Consider Example 5.6. Notice that every rule instantiation of the base rule, $T(x, y) \leftarrow R(x, y)$, is trivially minimal, useful and essential. As for the recursive rule, we

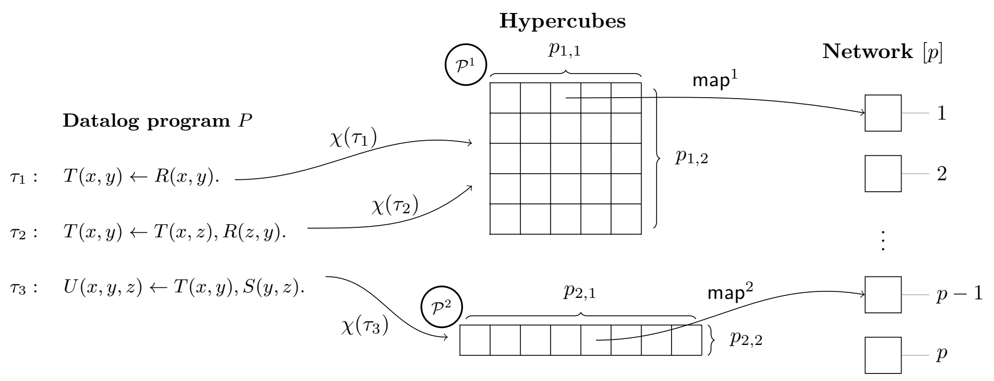

Figure 2 Example of a GHP policy for the Datalog program P with three rules.

showed in Example 5.6 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 [6]. Unfortunately, testing essentiality of a rule instantiation is undecidable. A proof of the proposition is given in Appendix B.7.

▶ **Proposition 5.11.** *Testing essentiality of a rule instantiation for a given Datalog program is undecidable.*

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 [2], which has been shown to provide nice guarantees on the communication-cost for CQ evaluation [7]. Let $P = \{\tau\}$ be a CQ with k distinct variables. Hypercube conceptually orders the p servers as a hypercube $\mathcal{P} = [p_1] \times [p_2] \times \cdots \times [p_k]$, 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 coordinate in space $\mathcal{P}$; and every variable x_i is associated with a hash function $h_{x_i} : \mathbf{dom} \mapsto [p_i]$. Then, a fact $R(a_1, \ldots, a_r)$, matching with atom $R(y_1, \ldots, y_r) \in \mathit{body}_\tau$, is sent to all servers whose coordinate agrees, for all $j \in [r]$, with position $h_{y_i}(a_j)$ on the dimension of $\mathcal{P}$ where y_j is associated with.

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"; for the specification of the production policy, we rely on a similar mechanism.

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* $\mathcal{P}^1, \ldots, \mathcal{P}^\ell$, for some parameter k.[3] The size of the dimensions of the hypercubes are parametrized by a matrix $\mathbf{P}$ of dimensions

[3] We assume w.l.o.g. that each hypercube has the same number of dimensions, but we can also define it such that different rules have a different number of dimensions.

$\ell \times k$, such that $\prod_{i=1}^{k} p_{j,i} = p$, for each $j \in [\ell]$. Each hypercube is then defined as $\mathcal{P}^j = [p_{j,1}] \times [p_{j,2}] \times \dots [p_{j,k}]$. For each hypercube $\mathcal{P}^j$, we also define a bijective mapping map^j that assigns to every point in $\mathcal{P}^j$ a machine $s \in [p]$. The latter thus provides the mapping between conceptual machines in the cube and real machines 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 $\rho^\tau : [k] \to 2^{vars(\tau)}$ that maps each dimension of the hypercube $\mathcal{P}^{\chi(\tau)}$ to a subset of the variables that appear in τ, such that the following condition holds:

If $\chi(\tau) = \chi(\tau')$, then $|\rho^\tau(i)| = |\rho^{\tau'}(i)|$ for every dimension i. In other words, the mappings of variables of different rules to the same hypercube must be consistent.

Finally, the GHP defines for each dimension $i \in [k]$ and each hypercube $\mathcal{P}^j$ a *hash function* h_i^j that maps sets of size α (α is the size of the set $\rho^\tau(i)$ for any τ such that $\chi(\tau) = j$) to a value in the i-th dimension. We require hash functions to be surjective and symmetric (by the latter we mean that $h_i^j(\boldsymbol{t}) = h_i^j(\boldsymbol{t'})$ if $\boldsymbol{t'}$ is a permutation of $\boldsymbol{t}$). Notice that our concept of hash-function is a generalization of the hash-functions used in, e.g., the Hypercube algorithm, where $\alpha = 1$. Further, we notice that, by definition, rules that use the same hypercube, also use the same hash functions 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 machines:

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

Intuitively, $S^\tau_{\boldsymbol{f},A}$ denotes the set of machines whose coordinate $\boldsymbol{q}$ is consistent with the for τ specified hash mappings. 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 $A \in body_\tau$ 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 $head_\tau$ with instantiation $\boldsymbol{f}$.

▶ **Example 6.1.** Consider the Datalog program depicted in Figure 2. We choose two hypercubes $\mathcal{P}^1, \mathcal{P}^2$ ($\ell = 2$) with dimension $k = 2$. The first two rules τ_1, τ_2 are mapped to the hypercube $\mathcal{P}^1$, and the third rule τ_3 is mapped to $\mathcal{P}^2$. 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 $\mathsf{map}^1, \mathsf{map}^2$ map the points of $\mathcal{P}^1, \mathcal{P}^2$ respectively to $\{1, \dots, p\}$ 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 4.3. Notice that since we use the same hypercube for both rules, the EDB 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 4.3 that broadcasts the EDB R to every machine 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 $\mathcal{P}^1$. For example, we can choose $p_{1,1} = p_{1,2} = \sqrt{p}$.

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

The following proposition shows that GHPs are strongly supporting policies.

▶ **Proposition 6.2.** *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.*

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* $\mathcal{H}$ is defined as the set of GHPs over P and $[p]$ that all have the same parametrization for $\mathbf{P}, \mathsf{map}^j, \chi, \rho^\tau$. In other words, policies in $\mathcal{H}$ 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 $\mathcal{F}_{\text{GHP}}$ 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 machines that derive a copy of the same fact? We start with a formal definition of boundedness.

▶ **Definition 7.1** (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 has reached a global fixpoint for $\boldsymbol{E}$ and P, when round k is finished. We call $\boldsymbol{E}$ ℓ-bounded if $k \leq \ell$.

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 7.2.** *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.*

Surprisingly, there exist economic policies for bounded Datalog programs that are not bounded. However, due to Proposition 7.2, such policies cannot be parallel-correct.

▶ **Example 7.3.** Consider the following bounded program.

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

We construct a network with $p > 1$ machines. Consider a policy that consumes $T(i)$ and $B(i)$ at machine $(i \mod p)+1$, and produces $T(i)$ at machine $(i \mod p)$. Every tuple in A is consumed at machine 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 in machine 1 at round 1, $T(1)$ is produced in machine 2 at round 2, and so on, until $T(p-1)$ is produced at round p at machine p.

In the remainder of this section, we focus on pure Datalog (denoted *PureDatalog*). We call a Datalog program *pure* if its variables occur at most once in every atom and it has no constants [23]. 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$_F(\mathcal{L}, \mathcal{W})$
Input:	Program $P \in \mathcal{L}$, family $\mathcal{F} \in \mathcal{W}$.
Question:	Is there a k s.t. $\mathcal{F}$ is k-bounded for P?

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

▶ **Theorem 7.4.** *1.* $\textsc{boundedness}_F(\mathit{PureDatalog}, \mathcal{F}_{\text{GHP}})$ *is undecidable;*
2. k-$\textsc{boundedness}(\mathit{PureDatalog}, \mathcal{E}_{\mathit{indep}})$ *and* k-$\textsc{boundedness}_F(\mathit{PureDatalog}, \mathcal{F}_{\text{GHP}})$ *are undecidable for* $k \geq 2$*; and*
3. k-$\textsc{boundedness}_F(\mathit{PureDatalog}, \mathcal{F}_{\text{GHP}})$ *is in* PTIME *if* $k = 1$*.*

Proofs for Theorem 7.4(1) and (2) are in the Appendix. Result (3) 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 (τ, v) exists with $v(\mathit{head}_\tau) = \boldsymbol{f}$, $v(\mathit{body}_\tau) \subseteq \mathsf{facts}_{\boldsymbol{C}}(s)$, and $v(\mathit{body}_\tau)$ 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 7.5.** *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* IDB *fact $\boldsymbol{f}$: (1) $|\boldsymbol{C}(\boldsymbol{f})| \leq 1$; and (2) $|\boldsymbol{C}(\boldsymbol{f})| = 1$ implies $\boldsymbol{C}(\boldsymbol{f}) = \boldsymbol{P}^*(\boldsymbol{f})$.*

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 $X \subseteq \mathit{vars}(A)$, we denote by $\mathit{pos}_A(X)$ the positions in A having variables from X.

▶ **Definition 7.6** (Pivoting Relation). A relation R is *pivoting w.r.t GHP family $\mathcal{H}$* if for every two atoms A_1 and A_2 (in rules τ_1 and τ_2 respectively) over R, and for all dimensions i of cube $\chi(\tau_1)$ with $p_{\chi(\tau_1),i} > 1$:
1. $\emptyset \subsetneq \rho^{\tau_1}(i) \subseteq \mathit{vars}(A_1)$; and
2. $\chi(\tau_1) = \chi(\tau_2)$ and $\mathit{pos}_{A_1}(\rho^{\tau_1}(i)) = \mathit{pos}_{A_2}(\rho^{\tau_2}(i))$.

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

▶ **Example 7.7.** For example, take the program

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

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 7.8** (Pivoting/Weakly pivoting). We say that a GHP family is *pivoting* (*weakly pivoting*, resp.) for P if all (all P-consumable, resp.) IDB relations are pivoting.

The program from Example 7.7 is weakly pivoting. We can test whether a GHP family is weakly pivoting in polynomial time, since we need to go over all P-consumable IDB relations, 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 7.9.** *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.*

We remark that Proposition 7.9 cannot be easily generalized. For example, one cannot replace GHP families by strongly supporting policies, since then, facts 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. 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 7.10.** 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 $\{1, \ldots, ar(R)\}$) onto 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 7.11** (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 $R \in \sigma$ and outputs a non-empty tuple with values in $[ar(R)]$. We say that $\boldsymbol{\beta}$ is a *pivot base* w.r.t. 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 w.r.t all relations in $\text{IDB}(P)$ (w.r.t all relations in $\text{IDB}(P)$ that occur in the body of some rule in P).

▶ **Example 7.12.** Consider the left-linear TC example, and let $\sigma = \{T\}$. Suppose we choose $\boldsymbol{\beta}(T) = (1)$. Then $\boldsymbol{\beta}$ is a pivot base w.r.t. 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 IDB 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 IDB relations, but only T occurs in the body of a rule. The pivot base $\boldsymbol{\beta}$ from before is still a pivot base w.r.t. $\{T\}$; hence the program is weakly pivoting. However, there is no pivot base w.r.t. to $\{T, U\}$, which means that the program is not pivoting.

The concept pivoting Datalog was first introduced in [30] for single rule programs and then generalized to full Datalog in [23] where it is called *generalized pivoting*. The latter definition is based on a rather complex argument over fractional weight-mappings, but relates to pivoting in that every generalized pivoting Datalog program is pivoting w.r.t. *all* IDB 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 7.13.** *Let P be a pure Datalog program and $p \geq 2$. There is a 1-bounded GHP family if and only if P is weakly pivoting.*

7.3 Bounded and Disjoint Evaluation

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

▶ **Definition 7.14** (Disjointness). Let P be a Datalog program, and R an IDB predicate 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. For this, 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* if $s \in \boldsymbol{C}(\boldsymbol{f})$ or $s \in \boldsymbol{P}^*(\boldsymbol{f})$ for all facts $\boldsymbol{f}$ of some IDB relation where P is defined over. Intuitively, a straggler is a server that consumes or produces an entire relation.

▶ **Proposition 7.15.** *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* IDB *relations, if and only if, $\mathcal{H}$ is pivoting.*

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

▶ **Proposition 7.16.** *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* IDB *relations.*

▶ Remark. The reader may wonder how the above concepts relate to the class of decomposable programs, as introduced in [32, 31]. 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 introduce 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 not undecidable. Another interesting direction for future work would be to define a relevant fairness condition for economic policies, e.g., an instance independent notion of load-balancing; and to study bounds on the amount of communication needed to evaluate Datalog programs. 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 Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

2 F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *EDBT '10*, pages 99–110, 2010.

3 Foto N. Afrati, Vinayak R. Borkar, Michael J. Carey, Neoklis Polyzotis, and Jeffrey D. Ullman. Map-reduce extensions and recursive queries. In *EDBT '11*, pages 1–8, 2011. URL: http://doi.acm.org/10.1145/1951365.1951367, doi:10.1145/1951365.1951367.

4 Foto N. Afrati and Christos H. Papadimitriou. The parallel complexity of simple chain queries. In *PODS '87*, pages 210–213, 1987. URL: http://doi.acm.org/10.1145/28659.28682, doi:10.1145/28659.28682.

5 Foto N. Afrati and Jeffrey D. Ullman. Transitive closure and recursive datalog implemented on clusters. In *EDBT '12*, pages 132–143, 2012. URL: http://doi.acm.org/10.1145/2247596.2247613, doi:10.1145/2247596.2247613.

6 Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and transferability for conjunctive queries. In *PODS '15*, pages 47–58. ACM, 2015. URL: http://doi.acm.org/10.1145/2745754.2745759, doi:10.1145/2745754.2745759.

7 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *PODS '13*, pages 273–284, 2013. URL: http://doi.acm.org/10.1145/2463664.2465224, doi:10.1145/2463664.2465224.

8 Paul Beame, Paraschos Koutris, and Dan Suciu. Skew in parallel query processing. In *PODS '14*, pages 212–223, 2014. URL: http://doi.acm.org/10.1145/2594538.2594558, doi:10.1145/2594538.2594558.

9 Shumo Chu, Magdalena Balazinska, and Dan Suciu. From theory to practice: Efficient join query evaluation in a parallel database system. In *SIGMOD '15*, pages 63–78, 2015. URL: http://doi.acm.org/10.1145/2723372.2750545, doi:10.1145/2723372.2750545.

10 S Cosmadakis and P Kanellakis. Parallel evaluation of recursive rule queries. In *PODS '86*, pages 280–293, New York, NY, USA, 1986. ACM. URL: http://doi.acm.org/10.1145/6012.15421, doi:10.1145/6012.15421.

11 J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI '04*, pages 137–150, 2004.

12 Hasanat M. Dewan, Salvatore J. Stolfo, Mauricio A. Hernández, and Jae-Jun Hwang. Predictive dynamic load balancing of parallel and distributed rule and query processing. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994.*, pages 277–288, 1994. URL: http://doi.acm.org/10.1145/191839.191893, doi:10.1145/191839.191893.

13 Sumit Ganguly, Abraham Silberschatz, and Shalom Tsur. Parallel bottom-up processing of datalog queries. *J. Log. Program.*, 14(1&2):101–126, 1992. URL: http://dx.doi.org/10.1016/0743-1066(92)90048-8, doi:10.1016/0743-1066(92)90048-8.

14 Sumit Ganguly, Avi Silberschatz, and Shalom Tsur. A framework for the parallel processing of datalog queries. In *SIGMOD '90*, pages 143–152, 1990. URL: http://doi.acm.org/10.1145/93597.98724, doi:10.1145/93597.98724.

15 Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and containment for conjunctive queries with union and negation. *CoRR*, abs/1512.06246, 2015. URL: http://arxiv.org/abs/1512.06246.

16 Hadoop. http://hadoop.apache.org/.

17 Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu, Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang, Andrew Whitaker,

Shengliang Xu, Magdalena Balazinska, Bill Howe, and Dan Suciu. Demonstration of the myria big data management service. In *SIGMOD '14*, pages 881–884, 2014. URL: `http://doi.acm.org/10.1145/2588555.2594530`, `doi:10.1145/2588555.2594530`.

18 Paris C. Kanellakis. *Logic programming and parallel complexity*, pages 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 1986. URL: `http://dx.doi.org/10.1007/3-540-17187-8_27`, `doi:10.1007/3-540-17187-8_27`.

19 Bas Ketsman and Dan Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *PODS '17*, pages 417–428, 2017. URL: `http://doi.acm.org/10.1145/3034786.3034788`, `doi:10.1145/3034786.3034788`.

20 Paraschos Koutris, Paul Beame, and Dan Suciu. Worst-case optimal algorithms for parallel query processing. In *ICDT '16*, pages 8:1–8:18, 2016. URL: `http://dx.doi.org/10.4230/LIPIcs.ICDT.2016.8`, `doi:10.4230/LIPIcs.ICDT.2016.8`.

21 Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In *PODS '11*, pages 223–234, 2011. URL: `http://doi.acm.org/10.1145/1989284.1989310`, `doi:10.1145/1989284.1989310`.

22 Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In *AAAI '14*, pages 129–137, 2014. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8505`.

23 Jürgen Seib and Georg Lausen. Parallelizing datalog programs by generalized pivoting. In *PODS '91*, pages 241–251, 1991. URL: `http://doi.acm.org/10.1145/113413.113435`, `doi:10.1145/113413.113435`.

24 Jiwon Seo, Jongsoo Park, Jaeho Shin, and Monica S. Lam. Distributed socialite: A datalog-based language for large-scale graph analysis. *PVLDB*, 6(14):1906–1917, 2013. URL: `http://www.vldb.org/pvldb/vol6/p1906-seo.pdf`.

25 Marianne Shaw, Paraschos Koutris, Bill Howe, and Dan Suciu. Optimizing large-scale semi-naïve datalog evaluation in hadoop. In *Datalog 2.0*, pages 165–176, 2012. URL: `http://dx.doi.org/10.1007/978-3-642-32925-8_17`, `doi:10.1007/978-3-642-32925-8_17`.

26 Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. Big data analytics with datalog queries on spark. In *SIGMOD '16*, pages 1135–1149, 2016. URL: `http://doi.acm.org/10.1145/2882903.2915229`, `doi:10.1145/2882903.2915229`.

27 Apache spark. `http://spark.apache.org/`.

28 Jeffrey D. Ullman and Allen Van Gelder. Parallel complexity of logical query programs. *Algorithmica*, 3:5–42, 1988. URL: `http://dx.doi.org/10.1007/BF01762108`, `doi:10.1007/BF01762108`.

29 Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. Asynchronous and fault-tolerant recursive datalog evaluation in shared-nothing engines. *PVLDB*, 8(12):1542–1553, 2015. URL: `http://www.vldb.org/pvldb/vol8/p1542-wang.pdf`.

30 O. Wolfson. Sharing the load of logic-program evaluation. In *DPDS '88*, pages 46–55, Dec 1988. `doi:10.1109/DPDS.1988.675001`.

31 Ouri Wolfson and Aya Ozeri. A new paradigm for parallel and distributed rule-processing. *SIGMOD Rec.*, 19(2):133–142, May 1990. URL: `http://doi.acm.org/10.1145/93605.98723`, `doi:10.1145/93605.98723`.

32 Ouri Wolfson and Avi Silberschatz. Distributed processing of logic programs. *SIGMOD Rec.*, 17(3):329–336, June 1988. URL: `http://doi.acm.org/10.1145/971701.50242`, `doi:10.1145/971701.50242`.

33 Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: SQL and rich analytics at scale. In *SIGMOD '13*, pages 13–24, 2013. URL: `http://doi.acm.org/10.1145/2463676.2465288`, `doi:10.1145/2463676.2465288`.

34 Weining Zhang, Ke Wang, and Siu-Cheung Chau. Data partition and parallel evaluation of datalog programs. *IEEE Trans. Knowl. Data Eng.*, 7(1):163–176, 1995. URL: `http://dx.doi.org/10.1109/69.368511`, `doi:10.1109/69.368511`.

A Proofs for Section 4

A.1 Proof for Proposition 4.4

▶ **Proposition 4.4.** *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 $L = \{\boldsymbol{g}_1, \dots, \boldsymbol{g}_k\} \subseteq children_T(\boldsymbol{g})$ be exactly those child nodes, with $round_T(\boldsymbol{g}_i) = M_{\boldsymbol{g}}$.

Now, we define $round_T(\boldsymbol{g}) = M_{\boldsymbol{g}}$ if $server_T(\boldsymbol{g}_i) = server_T(\boldsymbol{g})$, for all $\boldsymbol{g}_i \in L$, and $round_T(\boldsymbol{g}) = M_{\boldsymbol{g}} + 1$ 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 1.1.** *For Datalog program P, economic policy $\boldsymbol{E}$, instance I, and fact $\boldsymbol{f}$:*

1. *$\boldsymbol{f} \in [P, \boldsymbol{E}](I)$ implies existence of an annotated proof tree T for P, $\boldsymbol{E}$, I and $\boldsymbol{f}$. Specifically, 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 $round_{T'}(\boldsymbol{g}) \leq k$ (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 $\leq round_T(\boldsymbol{f})$. 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 4.4 is now a corollary of Lemma 1.1.

B Proofs for Section 5

B.1 Proof for Theorem 5.2

▶ **Theorem 5.2.** *PC(Datalog, $\mathcal{E}_{indep}$) is undecidable.*

Proof. The proof is by 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. As usual, we assume that both are over the same output predicate, 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 IDB predicates 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 $O(\boldsymbol{x}) \leftarrow O_i(\boldsymbol{x})$, 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 all O-facts to server 2. The EDB 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 $O(\boldsymbol{x}) \leftarrow O_1(\boldsymbol{x})$ is never used, since server 2 cannot consume facts over predicates 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 $O(I) = P_2(I)$ 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 every fact $O(\boldsymbol{a}) \in P(I)$ implies a fact $O_2(\boldsymbol{a}) \in P(I)$. The latter is equivalent to saying $O(\boldsymbol{a}) \in P_1(I)$ implies $O(\boldsymbol{a}) \in P_2(I)$ for every instance I, which means that $P_1 \subseteq P_2$. ◀

B.2 Proof for Lemma 5.3

▶ **Lemma 2.1.** *Let P be an arbitrary Datalog program and $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C}; U)$ an economic policy over σ that is parallel-correct for P. Now let $\boldsymbol{f} \in \mathsf{facts}(\sigma, U)$, and $\boldsymbol{C}'$ the consumption policy where $\boldsymbol{C}'(\boldsymbol{g}) = \boldsymbol{C}(\boldsymbol{g})$ for all $\boldsymbol{g} \in \mathsf{facts}(\sigma, U) \setminus \{\boldsymbol{f}\}$ and $\boldsymbol{C}'(\boldsymbol{f}) \subsetneq \boldsymbol{C}(\boldsymbol{f})$. It is still undecidable whether $\boldsymbol{E}'$ is parallel-correct for P.*

Proof. The proof is again by a reduction from the undecidable Datalog containment problem. Consider two Datalog programs P_1, P_2 with a single output relation (say T). We construct program P by considering again the indexed versions of P_1 and P_2, which we denote by P_1^* and P_2^*. (See the proof of Proposition 5.2). We further add also the following rules:

$$\begin{aligned} Trigger() &\leftarrow T_1(\boldsymbol{x}). \\ Transfer(\boldsymbol{x}) &\leftarrow Trigger(), T_1(\boldsymbol{x}). \\ T(\boldsymbol{x}) &\leftarrow Transfer(\boldsymbol{x}). \\ T(\boldsymbol{x}) &\leftarrow T_2(\boldsymbol{x}). \end{aligned}$$

We note that $edb(P) = edb(P_1^*) \cup edb(P_2^*)$ and $out(P) = \{T\}$. Consider the following economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ over a 2-node network: The consumption and production policies map every fact over relations with index 1 to server 1, and those with index 2 to 2. Additionally, $Trigger()$ is produced and consumed at server 1, all facts over $Transfer$ are in the production policy of server 1 and the consumption policy of server 2. Finally, all facts over T are in the production policy of server 2.

It is easy to see that program P is parallel-correct for $\boldsymbol{E}$. Indeed, program P_2 evaluates on server 2, where all tuples over T_2 are copied to T. Program P_1 evaluates on server 1, and if it generates tuples in T_1, then all are send to server 2 through the $Transfer$ relation. In server 2, all facts over $Transfer$ are copied to T.

Now let $\boldsymbol{C}'$ be the policy as described, but where $Trigger()$ is not consumed, *i.e.*, $\boldsymbol{C}'(Trigger()) = \emptyset$. We claim that the query implied by P_1 is contained in the query implied by P_2 if and only if economic policy $\boldsymbol{E}' = (\boldsymbol{P}, \boldsymbol{C})$ is parallel-correct for P.

(If). The proof is by contraposition. Let I be an instance for which $P_{1|\{T\}}(I) \not\subseteq P_{2|\{T\}}(I)$. It follows that $P_{|\{T\}}(I) \neq [P, \boldsymbol{E}](I)$. Indeed, in this case, there are tuples in relation ${T_1}^I$ that are not in ${T_2}^I$. Since $Trigger()$ is not consumed, these tuples will not be send to server 2 and thus missing in T. Hence, $\boldsymbol{E}'$ is indeed not parallel-correct for P.

(Only-if). Let I be an arbitrary instance. In the first round of $\boldsymbol{E}'$, all T_1 facts will be produced in machine 1, and all T_2 facts in machine 2. Moreover, all T facts that are produced from T_2 will also be produced in machine 2. It is easy to see that, since $P_{1|\{T\}}(I) \subseteq P_{2|\{T\}}(I)$ (notice that T here refers to the output relation of the original programs P_1, P_2, not the output of P), we have the complete result for T for P in machine 2. ◀

B.3 Proof for Proposition 5.5

We first show the following Lemma.

▶ **Lemma 2.2.** *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. We show that there exists a proof tree $T' \sqsubseteq T$ with depth $\leq d$ that uses only minimal rule instantiations.

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 EDB 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 entailed 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. ◀

▶ **Proposition 5.5.** *For every Datalog program P, we have $\mathcal{N}_P^{ess} \subseteq \mathcal{N}_P^{min} \cap \mathcal{N}_P^{use}$.*

Proof. 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 2.2, 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. ◀

B.4 Parallel Correctness Reformulation

We first show the following lemma.

▶ **Lemma 2.3.** *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 $children_T(root_T) \subseteq \mathsf{facts}_{\boldsymbol{C}}(i) \cap I \subseteq \mathsf{local}_i^1$ 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 say that an economic policy $\boldsymbol{E}$ *supports a proof tree* T if all the rule instantiations in T are supported.

▶ **Lemma 2.4.** *Let P be a Datalog program. An economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C}; U)$ is parallel-correct for P if and only if for every proof tree for P with fringe over $\mathsf{facts}(\sigma(P), U)$, an entailed 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 2.3 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 entailed 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. ◀

B.5 Proof for Proposition 5.7

▶ **Proposition 5.7.** *Let P be a Datalog program and $\boldsymbol{E}$ an economic policy. Then:*

1. *if $\boldsymbol{E}$ supports all minimal and useful rule instantiations in P, it is parallel-correct.*
2. *if $\boldsymbol{E}$ is parallel-correct for P, it supports all essential rule instantiations.*

Proof. The first item follows from Lemma 2.4 and Lemma 2.2. 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 Lemma 2.4, 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. ◀

B.6 Proof for Proposition 5.8

▶ **Proposition 5.8.** *Let P be a Datalog program where each* IDB *predicate 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 5.5 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. ◀

B.7 Proof for Proposition 5.11

We first show the following helper lemma.

▶ **Lemma 2.5.** *Testing whether for Datalog program P and rule $\tau \in P$ an essential rule instantiation exists is undecidable.*

Proof. We again consider 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 questions 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. ◀

▶ **Proposition 5.11.** *Testing essentiality of a rule instantiation for a given Datalog program is undecidable.*

Proof. The proof is by contradiction. We assume that testing essentiallity of a given rule instantiation is decidable and show that under this condition it is also decidable whether for a given rule an essential instantiation exists. Since the later contradicts with Lemma 2.5, the result follows.

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. Further, there are only finitely many distinct instantiations (up to isomorphisms).

For given rule $\tau \in P$, one can thus simply 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. ◀

C Proofs for Section 6

C.1 Proof for Proposition 6.2

▶ **Proposition 6.2.** *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.*

Proof. To show that $\boldsymbol{E}$ is 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 $S^{\tau}_{\boldsymbol{f},A}$, as defined in Section 6. 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 machine in $\bigcap_{A \in \tau} S^{\tau}_{A}$, which means that every instantiation of the rule τ will be strongly supported. ◀

D Proofs for Section 7

D.1 Proposition 7.2

▶ **Proposition 7.2.** *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 4.4. Boundedness of P implies that some constant exists, such that for every instance I and fact $\boldsymbol{f}$, $\boldsymbol{f} \in P(I)$ implies 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 monotonicity, and thus some proof-tree T with bounded fringe exists. It follows from (†) that $\boldsymbol{E}$ finished in a bounded number of rounds over fringe_T, and due to parallel-correctness of $\boldsymbol{E}$, $\boldsymbol{f} \in [P, \boldsymbol{E}](\mathit{fringe}_T)$.

Since this observation holds for all $\boldsymbol{f} \in [P, \boldsymbol{E}](I)$, it follows from Proposition 4.4 that $\boldsymbol{E}$ finishes in a bounded number of rounds.

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. ◀

D.2 Proof for Theorem 7.4

Results (1) and (2) follow from the bellow lemmas.

▶ **Lemma 4.1.** BOUNDEDNESS$_F$*(PureDatalog, $\mathcal{F}_{\text{GHP}}$) is undecidable.*

Proof. The proof is by reduction from the undecidable containment problem for Datalog programs. Let P_1, P_2 be two Datalog programs with same distinguished output predicate that serve 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 schema $\sigma(P_1^*) \cup \sigma(P_2^*) \cup \{\text{Adom}^{(1)}, T^{(2+ar(O))}, E^{(2)}\}$ by combining the rules from P_1^*, P_2^*, and those mentioned below. First we add rules $\mathit{Adom}(x_j) \leftarrow X(x_1, \ldots, x_\alpha)$ for every relation $X^{(\alpha)} \in \sigma(P_1^*) \cup \sigma(P_2^*) \cup \{E^{(2)}\}$ and $j \in [\alpha]$. Further, we add:

$$T(\boldsymbol{z}; x, y) \leftarrow O_1(\boldsymbol{z}), \mathit{Adom}(x), \mathit{Adom}(y). \tag{τ_1}$$

$$T(\boldsymbol{z}; x, y) \leftarrow O_2(\boldsymbol{z}), E(x, y). \tag{τ_2}$$

$$T(\boldsymbol{z}; x, y) \leftarrow T(\boldsymbol{z}; x, w), T(\boldsymbol{z}; w, y). \tag{τ_3}$$

Notice that new relation E is an EDB relation, while T and Adom are IDB relations.

Next, we define a GHP $\mathcal{H}$. For this take a single 1-dimensional cube of p 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 τ_1 we again define $\rho^{\tau_1}(1) = \emptyset$, for τ_2 and τ_3 we define $\rho^{\tau_2}(1) = \rho^{\tau_3}(1) = \{x, y\}$.

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, program P_1^* and P_2^*, as well as relation *Adom* are fully computed locally on every server. Further, during this same round every T-fact with prefix a tuple from O_2 is computed. After the first round, several relations will be communicated: $\boldsymbol{C}$-consumable relations used by P_1^* and P_2^*, as well as relation T and Adom. Since all these relations where computed on all 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 $O(\boldsymbol{t}) \in P_1(I')$, $O(\boldsymbol{t}) \notin P_2(I')$. We convert instance I to an instance for P, by annotating the relations with respective index, and add a relation E with chain $E(0,1), E(1,2), \ldots, E(m-1, m)$, for some integer m.

We take a specific GHP from $\mathcal{H}$. For this we need some additional notation. Let $\mathcal{T} = \{T(\boldsymbol{t}, \boldsymbol{u}) \mid \boldsymbol{t} \in \mathit{adom}(I)^k, \boldsymbol{u} \in \mathit{TC}(E^I)\}$. Intuitively, $\mathcal{T}$ contains all T facts that can be derived based on the given E relation, for all possible prefixes.

Not let π be the function where $\pi(1) = 1$ and $\pi(i) = j$, with $2^{j-1} < i \leq 2^j$. By B_j we denote the subset of $\mathcal{T}$ representing chains of length i, for all $\pi(i) = j$. Then as hash function we choose $h_1^1(\{i, j\}) = (\pi(i+j) \bmod p) + 1$.

During the computation of P over I we observe that P_1^*, P_2^*, and Adom are computed in parallel on all available server as before. Only now, there are facts $O_2(\boldsymbol{a})$ such that $O_1(\boldsymbol{a})$ does not exist, more specifically, T-facts with prefix $\boldsymbol{t}$. Due to choice of hash functions, these are all produced (only) on server 1. Hence, after the consequent communication phase, and due to the construction of I, the production of new facts will be triggered.

More precisely we have the following induction hypothesis: After communication round $i+1$ all servers know all T facts in $B_j \cap P(I)$, for all $j \leq i$, and no server has produced facts from B_j, with $j > i$. Further, exactly one server is able to produce new T facts based on the facts in $B_i \cap P(I)$, and this server is capable of computing only the facts in $B_{i+1} \cap P(I)$.

For round i we observe that initially, all servers know all T-facts from $B_j \cap P(I), j \leq i$, due to the induction hypothesis. We observe that all new T-relations that can be obtained by applying τ_3 on $B_j \cap P(I), j \leq i$ are in B_{i+1}, and thus can be produced only by server $(i \bmod p) + 1$. Since the production of facts in B_{i+2} is assigned to another server (by choice of hash function), during this round only the facts from $B_{i+1} \cap P(I)$ can be produced.

Since the number of buckets B_i is logarithmic in the size of E, it follows that to reach a fixpoint, we need $\mathcal{O}(\log(m))$ rounds. ◀

▶ **Lemma 4.2.** *k-*BOUNDEDNESS$_F$*(PureDatalog, $\mathcal{F}_{\text{GHP}}$) is undecidable if $k \geq 2$.*

Proof. We give a reduction from the undecidable Datalog containment problem. Given two Datalog programs P_0, P_1 with single output predicate T, which serve as input for Datalog containment. First, we modify program P_0 and P_1 in the following way. For each EDB relation $R^{(\alpha)} \in \mathit{edb}(P_i)$ and every position $j \in [\alpha]$, we add the rule $\mathit{Adom}_i(x_j) \leftarrow R(x_1, \ldots, x_\alpha)$ to program P_i. Then, we add to every relation in $\mathit{idb}(P_i)$ (expect for newly introduced relation Adom_i) one additional attribute. For notational convenience, we assume this is the first attribute of each relation. Then we update all the rules in P_i, by adding to every IDB (applicable) predicate an additional variable z on the first position, and add atom $Atom_i(z)$ to the body of the rule. Here we assume that z is a fresh variable, not previously used by the rule. We refer to the annotated programs as P_0^* and P_1^*, and by $T_0^{(\beta)}$ and $T_1^{(\beta)}$ as their respective output predicates. Notice that P_0^* and P_1^* are essentially equivalent to P_0 and P_1, but facts are computed redundantly for every value in the considered instance.

We construct program P over schema $\sigma(P_0^*) \cup \sigma(P_1^*) \cup \{T_i^{(\beta)} \mid i \in [k-1]\}$ by combining the rules in P_0^* and P_1^* with the below rules, for $i \in [k-1]$.

$$T_i(z; \boldsymbol{x}) \leftarrow T_{i-1}(z; \boldsymbol{x}), Adom_{(i+1 \bmod 2)}(z).$$

Next, we construct a GHP family $\mathcal{H}$ over a p-server network (with $p \geq 2$) that considers two 1-dimensional cube configurations of the p servers, each with their own hash function. We refer to these cubes as *cube* 0 and *cube* 1. We assume the identity mapping map from cube coordinates to $[p]$ that is the same for both cubes. Now every rule having a head predicate with index i is mapped over cube (i mod 2) by applying the associated hash function over variable z. We refer to these hash functions as h_0 and h_1 respectively.

Now, we show that $\boldsymbol{E}$ is k-bounded if and only if $P_0 \subseteq P_1$.

If. On every instance I, by construction of $\boldsymbol{E}$, program P_1^* and P_2^* will be computed completely already after a single round. This follows from the fact that P_1^* and P_2^* are pivoting and $\boldsymbol{E}$ decomposes their computation.

In the consequent communication phase (i.e., round 2), all servers send the tuples from their locally computed T_i relation to the respective consuming servers. In the computation phase that follows, servers receiving fact $T_0(a; \boldsymbol{t})$ derive fact $T_1(a; \boldsymbol{t})$. Since the former is consumed at server $f(h_1(a))$, and the latter was already produced at server $f(h_1(a))$ during the previous round (due to $P_0 \subseteq P_1$ and the constructed GHP family), these need not be communicated.

For $k = 2$, we have thus reached the fixpoint, particularly because then relation T_1 is not consumed. For $k > 2$, we observe that in each additional communication round i (with $k \geq i > 2$), only tuples with predicate T_j (with $j \geq i-1$) are reshuffled and copied into relation T_{j+1}. In a worst-case scenario we thus need k rounds for all these facts to be copied to relation T_{k-1}.

Only if. Suppose $P_1 \not\subseteq P_2$. We fix some arbitrary hash function for h_0 over $[p]$ and define h_1 as the hash function over $[p]$ where $h_1(a) = h_0(a) + 1 \bmod p$, for every value $a \in \mathbf{dom}$. Notice that h_0 and h_1 yield a GHP in $\mathcal{H}$.

Let $T(\boldsymbol{t})$ be a fact and I and instance with $T(\boldsymbol{t}) \in P_0(I) \backslash P_1(I)$. Notice that in program P, this fact has the form $T_0(a; \boldsymbol{t})$, for some value a in the active domain of I. The computation of I proceeds as in the previous case, only after the first communication phase some server receives fact $T_0(a; \boldsymbol{t})$, deriving a fact $T_1(a; \boldsymbol{t})$ that was not previously in the T_1 relation (recall that by assumption P_2 could not derive $T_2(a; \boldsymbol{t})$). This fact will thus be send to its consumers in communication phase 3 (rather than 2).

Due to our choice of hash functions, facts from relation T_i are communicated the earliest in round i, and the fact $T_i(a; \boldsymbol{t})$, for $i \geq 2$, is being communicated in the $i{+}1$th communication phase. Therefore $k+1$ rounds are needed before fact $T_{k+1}(a; \boldsymbol{t})$ is derived. ◀

▶ **Lemma 4.3.** *k-BOUNDEDNESS$(PureDatalog, \mathcal{E}_{indep})$ is undecidable if $k \geq 2$.*

Proof. The proof is 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 T_1 is the output predicate for P_1, and T_2 for P_2. Furthermore we add the following rules:

$T_2(\boldsymbol{x}) \leftarrow T_1(\boldsymbol{x})$.
$O(\boldsymbol{x}) \leftarrow T_2(\boldsymbol{x})$.

We take as economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ over a two-node network. The consumption and production policies map every fact over relations with index 1 to server 1, and those with index 2 to server 2. Additionally, relation T_1 is also consumed at server 2, and relation T_2 is also consumed at server 1. Relation O is produced at server 1. Next, we show that $\boldsymbol{E}$ is 2-bounded if and only if $P_1 \subseteq P_2$.

(If). On every instance I, server 1 computes the complete output $P_1(I)$, and server 2 computes the complete output $P_2(I)$. In the consequent communication phase, server 1 sends its entire T_1 relation to server 2, and server 2 sends its entire T_2 relation to server 1. In the computation phase that follows, server 2 adds all the received facts from T_1 to its relation T_2, which has no effect due to assumption $P_1 \subseteq P_2$. Hence server 2 requires no additional communication and reached a local fixedpoint. Server 1 copies relation T_2 to relation O. As no other new facts are derived than those in O, and relation O is not consumed, server 1 has reached a local fixed too. Hence, $\boldsymbol{E}$ is indeed 2-bounded.

(Only if). Suppose $P_1 \not\subseteq P_2$. Let $\boldsymbol{f}$ be a fact and I and instance with $\boldsymbol{f} \in P_1(I) \setminus P_2(I)$. The computation of I proceeds as in the previous case, only after the first communication phase, server 2 receive at least one fact ($\boldsymbol{f}$) that was not computed locally. In the consequent computation phase, this fact will be added to the T_2 relation, and needs to be communicated to server 1 before a fixpoint can be reached. Policy $\boldsymbol{E}$ is thus not 2-bounded. ◀

D.3 Proof for Proposition 7.5

▶ **Proposition 7.5.** *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* IDB *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 IDB facts derived during the distributed evaluation are P-derivable. Consider a rule instantiation (τ, v) that is satisfied 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 EDB facts are never communicated after round 1.

(Only if). Towards a contradiction, suppose that $\boldsymbol{E}$ is 1-bounded, but the conditions of the proposition do not hold, *i.e.*, there exists some P-derivable fact $\boldsymbol{f}$ such that at least one of conditions (1), (2) fails. Because $\boldsymbol{E}$ is strongly supporting, it is parallel-correct and thus for some instance I and server s, during the distributed evaluation, server s will produce fact $\boldsymbol{f}$ in round 1. Now, if condition (1) is not true, then $|\boldsymbol{C}(\boldsymbol{f})| > 1$, which implies that server s needs to send $\boldsymbol{f}$ to servers $\boldsymbol{C}(\boldsymbol{f}) \setminus \{s\}$; this contradicts 1-boundedness. Similarly, if condition (1) holds but (2) does not, then $s \in \boldsymbol{P}^*(\boldsymbol{f})$, while for some server $s' \neq s$ we have $\boldsymbol{C}(\boldsymbol{f}) = \{s'\}$. By the definition of $\boldsymbol{P}^*$, there exists some instance J, rule τ and valuation v, with $v(head_\tau) = \boldsymbol{f}$, such that all facts $v(body_\tau)$ are derived and are consumed on server s. This means that $\boldsymbol{f}$ will be produced (during round 1), so then it will have to be sent to s'; this contradicts again 1-boundedness. ◀

D.4 Proof for Proposition 7.9

▶ **Proposition 7.9.** *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 $\mathcal{H}$. Let $\boldsymbol{f}$ be an arbitrary $\boldsymbol{C}$-consumable fact over the schema of P. Recall that $s \in \boldsymbol{C}(\boldsymbol{f})$ iff there is a $\tau \in P, A \in body_\tau$, and valuation v such that $v(A) = \boldsymbol{f}$. Analogously, $s \in \boldsymbol{P}(\boldsymbol{f})$ iff there is a $\tau \in P$ and valuation v such that $v(A) = \boldsymbol{f}$, with $A = head_\tau$.

More precisely, in both cases, for fixed τ and A, condition (1) of weakly pivoting GHP families implies that server s is uniquely identified by 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 $\chi(\tau)$.)

Clearly, $\boldsymbol{C}(\boldsymbol{f}) \neq \emptyset$, because $\boldsymbol{f}$ is over a P-consumable relation, and $\boldsymbol{E}$ is a GHP. Now take $s_1, s_2 \in \boldsymbol{C}(\boldsymbol{f})$. Then, due to condition (2) of weakly pivoting GHPs it follows directly that $s_1 = s_2$. Hence, $|\boldsymbol{C}(\boldsymbol{f})| \leq 1$.

For $s_1 \in \boldsymbol{P}(\boldsymbol{f})$ and $s_2 \in \boldsymbol{C}(\boldsymbol{f})$ the observation that $s_1 = s_2$ is analogous. Hence, $\boldsymbol{P}(\boldsymbol{f}) = \boldsymbol{C}(\boldsymbol{f})$. Now it follows from Proposition 7.5 that $\boldsymbol{E}$ is 1-bounded.

(Only If). We argue condition (1) and (2) from the definition of weakly pivoting GHPs by contraposition. First suppose that (1) fails for some τ_1, i, and A_1. If A_1 is a body atom it follows immediately that $\boldsymbol{f}$ is replicated for $\boldsymbol{C}$ over dimension i, which contradicts 1-boundedness (since $p_i > 1$). 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 facts matching the head of τ_1. Since R_1 is $\boldsymbol{C}$-consumable and $p_i > 1$, this again contradicts 1-boundedness. For the case where $\rho^{\tau_1}(i) \neq \emptyset$, a similar argument holds: Take $x \in \rho^{\tau_1}(i) \setminus vars(A_1)$ 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 satisfy on distinct 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 satisfy on distinct servers. Since all matching facts are $\boldsymbol{C}$-consumable, this would contradict 1-boundedness. For $\chi(\tau_1) = \chi(\tau_2)$ and $pos_{A_1}(\rho^{\tau_1}(i)) \neq pos_{A_2}(\rho^{\tau_2}(i))$ the observation is analogous. ◀

D.5 Proof for Proposition 7.13

▶ **Proposition 7.13.** *Let P be a pure Datalog program and $p \geq 2$. There is a 1-bounded GHP family if and only if P is weakly pivoting.*

In the below propositions, we show slightly stronger results. The if-direction of Proposition 7.13 then follows from Proposition 7.9 and Proposition 4.4. The only-if direction follows from Proposition 4.5. Proposition 4.6.

▶ **Proposition 4.4.** *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 B 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. ◀

▶ **Proposition 4.5.** *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* IDB *relations.*

Proof. To show that $\boldsymbol{E}$ is without stragglers, we recall that the hash functions used by $\boldsymbol{E}$ are surjective by definition. Thus for any P-consumable relation R a fact $\boldsymbol{f}$, a fact $\boldsymbol{g}$ over R with $\boldsymbol{C}(\boldsymbol{f}) \neq \boldsymbol{C}(\boldsymbol{g})$ can always be found. ◀

▶ **Proposition 4.6.** *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* IDB *relation in P.*

In the remainder of this section, we prove Proposition 4.6. We introduce the notion of policy key for economic policy $\boldsymbol{E} = (\boldsymbol{C}, \boldsymbol{P})$ and Datalog program P. Let R be some IDB 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, $\boldsymbol{f}[\gamma] = \boldsymbol{g}[\gamma]$ 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 $\boldsymbol{C}(\boldsymbol{f}) = \boldsymbol{C}(\boldsymbol{g}) = \{s\}$. 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 IDB relations have a (possibly empty) policy key, which follows immediately from Proposition 7.5.

▶ **Lemma 4.7.** *For pure Datalog program P, and 1-bounded strongly supporting economic policy $\boldsymbol{E} = (\boldsymbol{P}, \boldsymbol{C})$ for P, the following are equivalent:*

1. *a $\boldsymbol{C}$-consumable* IDB *relation of P has empty key in $\boldsymbol{E}$;*
2. *$\boldsymbol{E}$ has a straggler for some $\boldsymbol{C}$-consumable* IDB *relation.*

Proof. The direction (2) $\Rightarrow$ (1) is straightforward. For (1) $\Rightarrow$ (2), let R be a $\boldsymbol{C}$-consumable IDB relation with empty key. The latter means that some server s exists with $s \in \boldsymbol{C}(\boldsymbol{f})$ for all facts $\boldsymbol{f}$ with predicate R. (Recall that $s \in \boldsymbol{P}^*(\boldsymbol{f})$ implies $s \in \boldsymbol{C}(\boldsymbol{f})$ due to Proposition 7.5 and the fact that R is $\boldsymbol{C}$-consumable.) ◀

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

▶ **Lemma 4.8.** *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 $\boldsymbol{f}[\gamma] = \boldsymbol{g}[\gamma]$.

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 4.9.** *Let P be a pure Datalog program and $\boldsymbol{E}$ a strongly supporting economic policy. Let $R_1(\boldsymbol{x_1})$, $R_2(\boldsymbol{x_2})$ be two* IDB *relations occurring in some rule $\tau \in P$, with minimal keys $\boldsymbol{\gamma_1}$ and $\boldsymbol{\gamma_2}$ 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. We show that every tuple $\boldsymbol{\gamma}$ consisting of all positions in $pos_{A_1}(vars_{A_1}[\boldsymbol{\gamma_1}] \cap vars_{A_2}[\boldsymbol{\gamma_2}])$ 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 4.8.

Now let $\boldsymbol{f}, \boldsymbol{g}$ be two arbitrarily chosen facts over R_1, with $\boldsymbol{f}[\gamma] = \boldsymbol{g}[\gamma]$. Let v be a valuation for τ, with $v(A_1) = \boldsymbol{f}$; and let v' be a valuation for τ with $v'(A_1) = \boldsymbol{g}$. Then, we construct valuation v'' so that $v''(x) = v(x)$ for all $x \in vars(A_2)$; and $v''(x) = v'(x)$ for all $x \in vars(A_1) \setminus vars(A_2)$.

We observe that $v''(A_2) = v'(A_2)$, and thus trivially $\boldsymbol{C}(v''(A_2)) = \boldsymbol{C}(v'(A_2)) = \{s\}$. The latter equality is due to existence of a key for R_2. More precisely, all P-consumable body facts from instantiations (v', τ) and (v'', τ) must be consumed on server s due to $\boldsymbol{E}$ being strongly supporting. If R_1 is P-consumable, this implies $s \in \boldsymbol{C}(\boldsymbol{f}) \cap \boldsymbol{C}(v'(A_2))$. Due to existence of a key for R_1, $|\boldsymbol{C}(\boldsymbol{f})| = 1$, thus $\boldsymbol{P}^*(\boldsymbol{f}) = \boldsymbol{C}(\boldsymbol{f}) = \{s\}$. Analogously we obtain $\boldsymbol{P}^*(\boldsymbol{g}) = \boldsymbol{C}(\boldsymbol{g}) = \{s\}$, which is as desired.

If A_1 is the head of τ, then $s \in \boldsymbol{P}^*(\boldsymbol{f}) \cap \boldsymbol{C}(v'(A_1))$. Due to existence of a key for R_1, either $\boldsymbol{P}^*(\boldsymbol{f}) = \boldsymbol{C}(\boldsymbol{f}) = \{s\}$ or $\boldsymbol{C}(\boldsymbol{f}) = \emptyset$. As before, an analogous observation gives $\boldsymbol{P}^*(\boldsymbol{g}) = \{s\}$ and $\boldsymbol{C}(\boldsymbol{g}) = \boldsymbol{P}(\boldsymbol{g}) = \{s\}$, or $\boldsymbol{C}(\boldsymbol{g}) = \emptyset$. Due to pureness either both facts are P-consumable and thus $\boldsymbol{C}$-consumable s, or not. Hence the result is as desired. ◀

Proof of Proposition 4.6. 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 IDB relations. Then, by Lemma 4.7 all P-consumable relations have non-empty policy keys. In turn, this implies that from Lemma 4.9 we can use the policy keys for $\boldsymbol{C}$-consumable IDB relations as pivot base. However, this contradicts the fact that P is not weakly pivoting. ◀

D.6 Proof for Proposition 7.15

▶ **Proposition 7.15.** *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* IDB *relations, if and only if, $\mathcal{H}$ is pivoting.*

The result follows from the below two propositions: Proposition 4.10 and Proposition 4.11.

▶ **Proposition 4.10.** *Let $\boldsymbol{E}$ be a pivoting GHP for some pure Datalog program P. Then $\boldsymbol{E}$ is 1-bounded, disjoint, and without stragglers for* IDB *relations.*

Proof. Since a pivoting GHP is also weakly pivoting, it follows from Proposition 7.9 and Proposition 4.5 that $\boldsymbol{E}$ is 1-bounded and without stragglers for P-consumable IDB relations.

For the remainder of the proof we observe that $s \in \boldsymbol{P}^*(\boldsymbol{f})$ iff there is a rule $\tau \in P$ and valuation v such that $v(head_\tau) = \boldsymbol{f}$. Due to condition (1) of pivoting GHP, 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 from condition (2) of pivoting GHP, 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 IDB relations. ◀

▶ **Proposition 4.11.** *Let P be a pure Datalog program, and $\mathcal{H}$ a GHP family. If $\mathcal{H}$ is 1-bounded, disjoint for P without stragglers for* IDB *relations, then $\mathcal{H}$ is pivoting.*

For the proof, we use the next auxiliary result.

▶ **Proposition 4.12.** *An economic policy $\boldsymbol{E} = (\boldsymbol{C}, \boldsymbol{P})$ that is 1-bounded, disjoint, strongly supporting and without stragglers has non-empty keys for* IDB *relations of the associated Datalog program.*

Proof. Due to 1-boundedness and the absence of stragglers for P-consumable IDB relations, it follows from Lemma 4.7 that non-empty keys for $\boldsymbol{C}$-consumable IDB relations exist.

Now let R be an IDB relation that is not P-consumable. Pureness of P, and $\boldsymbol{E}$ being disjoint and strongly supporting implies $|\boldsymbol{P}^*(\boldsymbol{f})| = 1$, thus R has a trivial key. Due to the absence of stragglers it immediately follows that R cannot have the empty key. ◀

Proof for Theorem 4.11. 1-boundedness implies that $\mathcal{H}$ is weakly pivoting due to Proposition 4.12. 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 7.9. 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. ◀

D.7 Proof for Proposition 7.16

▶ **Proposition 7.16.** *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* IDB *relations.*

Proof. *(If).* The proof is analogous to the proof of Proposition 4.4, by taking the pivot base B for P to obtain a pivoting GHP for P. The result then follows from Proposition 4.10.

(Only if). The result follows from Proposition 4.12. Particularly, because we can take the non-empty keys for IDB relations as pivot. Correctness follows from Correctness follows from non-emptyness of policy keys and Lemma 4.9. ◀