Polynomial Time Convergence of the Iterative Evaluation of Datalogo Programs

SUNGJIN IM, University of California at Santa Cruz, USA
BENJAMIN MOSELEY, Carnegie Mellon University, United States
HUNG Q. NGO, RelationalAI, USA
KIRK PRUHS, University of Pittsburgh, USA

Datalog° is an extension of Datalog that allows for aggregation and recursion over an arbitrary *commutative semiring*. Like Datalog, Datalog° programs can be evaluated via the natural iterative algorithm until a fixed point is reached. However unlike Datalog, the natural iterative evaluation of some Datalog° programs over some semirings may not converge. It is known that the commutative semirings for which the iterative evaluation of Datalog° programs is guaranteed to converge are exactly those semirings that are stable [11]. Previously, the best known upper bound on the number of iterations until convergence over p-stable semirings is $\sum_{i=1}^{n}(p+2)^i = \Theta(p^n)$ steps, where n is (essentially) the output size. We establish that, in fact, the natural iterative evaluation of a Datalog° program over a p-stable semiring converges within a polynomial number of iterations. In particular our upper bound is $O(\sigma p n^2(n^2 \lg \lambda + \lg \sigma))$ where σ is the number of elements in the semiring present in either the input databases or the Datalog° program, and λ is the maximum number of terms in any product in the Datalog° program.

CCS Concepts: • **Theory of computation → Database query languages (principles)**.

Additional Key Words and Phrases: Datalog, convergence time, semiring

ACM Reference Format:
Sungjin Im, Benjamin Moseley, Hung Q. Ngo, and Kirk Pruhs. 2024. Polynomial Time Convergence of the Iterative Evaluation of Datalogo Programs. *Proc. ACM Manag. Data* 2, 5 (PODS), Article 221 (November 2024), 19 pages. https://doi.org/10.1145/3695839

1 Introduction

Motivated by the need in modern data analytics to express recursive computations with aggregates, Khamis et al. [11] introduced Datalog°, which is an extension of Datalog that allows for aggregation and recursion over an arbitrary *commutative semiring*.[1] As a quick example, given a graph with edge set E, the Datalog rule for computing the transitive closure of E can be written as (see [1, 3]):

$$T(X,Y) \text{ :- } E(X,Y) \vee \exists_Z \ (T(X,Z) \wedge E(Z,Y)) \tag{1}$$

[1]The results in [11] are on *Partially Ordered Pre-Semirings* (POPS). However, the key convergence properties are reflected in the *core semiring* of the POPS. Thus, it is sufficient to restrict our attention to semirings for the purpose of this paper.

Authors' Contact Information: Sungjin Im, sungjin@ucsc.edu, University of California at Santa Cruz, Computer Science and Engineering, Santa Cruz, California, USA; Benjamin Moseley, Carnegie Mellon University, Tepper School of Business, Pittsburgh, PA, United States, moseleyb@andrew.cmu.edu; Hung Q. Ngo, RelationalAI, Berkeley, CA, USA, hung.q.ngo@gmail.com; Kirk Pruhs, University of Pittsburgh, Computer Science Department, Pittsburgh, PA, USA, kirk@cs.pitt.edu.

ACM 2836-6573/2024/11-ART221
https://doi.org/10.1145/3695839

Datalog$^\circ$ expands the expressiveness of the above rule by replacing $\vee$ with $\oplus$ and $\wedge$ with $\otimes$, for a given semiring[2] $\boldsymbol{S} = (S, \oplus, \otimes, 0, 1)$; the rule becomes:

$$T(X, Y) \text{ :- } E(X, Y) \oplus \bigoplus_Z (T(X, Z) \otimes E(Z, Y)), \tag{2}$$

where T and E are no longer just relations, but they are functions $\mathsf{Dom} \times \mathsf{Dom} \to S$, where Dom is the domain of X, Y and Z. These functions are also called $\boldsymbol{S}$-relation, or K-relation in [8]. The semantic and interpretation of the rule depends on the semiring $\boldsymbol{S}$ and the database instance (defining the function E) at hand. For example, when $\boldsymbol{S}$ is the Boolean semiring we get back the transitive closure problem; when $\boldsymbol{S}$ is the min-plus semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$, and E is the edge-weight function of a graph, the rule defines recursively the shortest path length between every node pair in a graph[3]

$$T(X, Y) \text{ :- } \min\{E(X, Y), \min_Z (T(X, Z) + E(Z, Y))\}. \tag{3}$$

The striking similarity between transitive closure and all pairs shortest paths and a host of other problems was recognized since at least the 1960s [6, 14, 19, 21, 22]. This is often referred to as the *algebraic path problem* [18]. Standard algorithms textbooks (e.g. [3]) present generic solutions to this problem, such as the Floyd-Warshall algorithm.

In our much more general setting, the transitive closure rule becomes Datalog, and the algebraic path problem becomes Datalog$^\circ$. Like Datalog, Datalog$^\circ$ programs can be evaluated via the natural iterative algorithm until a fixed point is reached. This is sometimes called the "naïve evaluation" algorithm, or in other contexts Jacobi iteration, Gauss-Seidel iteration, or Kleene iteration [7, 18]. Furthermore, Datalog$^\circ$ is attractive for practical applications because it also allows for a generalization of semi-naïve evaluation to work, under some assumptions about the semiring [11]. While semi-naïve evaluation makes each iteration faster to compute, the total number of iterations is the same as that of the naïve evaluation algorithm. Thus, bounding the number of iterations of the naïve evaluation is an important question in practice.

In Datalog, it is easy to see that the number of iterations until a fixed point is reached is at most the output size. This is because every iteration before convergence must derive at least one new fact, due to monotonicity. For example, the transitive closure rule (1) reaches a fixpoint if we can no longer infer the reachability of a new pair (X, Y) of vertices. In contrast, the naïve evaluation of Datalog$^\circ$ programs over some commutative semirings may not converge. A simple example is the sum-product semiring over the reals.

It is known that the commutative semirings for which the iterative evaluation of Datalog$^\circ$ programs is guaranteed to converge are exactly those semirings that are stable [11]. A semiring is *p-stable* [7] if $f^{(p+1)}(0) = f^{(p)}(0)$ for every linear function $f(x) = a \otimes x \oplus b$ over the semiring. (See Section 2 for a more formal definition of stability.) A semiring is stable if there exists a p for which it is p-stable. Previously, the best known upper bound [11] on the number of iterations until convergence is $\sum_{i=1}^{n} (p+2)^i = \Theta((p+2)^n)$ steps, where n is (essentially) the output size, and p is the stability index of the underlying semiring. In contrast there are no known lower bounds that show that iterative evaluation requires an exponential (in the parameter n) number of steps to reach convergence.

There are special cases where polynomial convergence rate is known. The first case is when the semiring is 0-stable, as in the standard Boolean semiring, where it is known that naïve evaluation converges in $O(n)$ steps [11]. The second case is when the input program is *linear*, meaning that in every rule the product only contains at most one IDB relational symbol. In [10] it is shown that if

[2]See Section 2 for a formal definition of semirings.

[3]Here, we wrote min in prefix notation for readability. The generic $\oplus$ operator is more naturally written in infix notation.

the semiring is p-stable with L elements in the semiring domain, then the iterative evaluation of all linear Datalog$^\circ$ programs converge after $O(\min(pn^3, pn \lg L))$ steps.

Our Contributions. The open problem we address in this paper is whether the iterative evaluation of Datalog$^\circ$ programs over p-stable semirings might indeed require an exponential number of steps to converge. Another way to frame our motivating research question is whether or not polynomial convergence is a special property of *linear* Datalog$^\circ$ programs that is not shared by general Datalog$^\circ$ programs. Our main finding is stated in Theorem 1.1.

Theorem 1.1. *Let $\mathbf{S}$ be a p-stable commutative semiring. Let P be a* Datalog$^\circ$ *program where the maximum number of multiplicands in any product is at most λ. Let D be the input database instance. Let σ be the number of the semiring elements referenced in P or D. Let n denote the total number of ground atoms in an IDB that at some point in the iterative evaluation of P over $\mathbf{S}$ on input D have a nonzero associated semiring value. Then the iterative evaluation of P over $\mathbf{S}$ on input D converges within*

$$\lceil pn(n+3) \cdot (\sigma(n(n+3)/2) \lg(\lambda+1) + 4\sigma \lg \sigma + 1) \rceil$$

steps.

As λ is only a property of the Datalog$^\circ$ program, and p is only a property of the semiring, they do not scale with the data size. Thinking of them as constants in data complexity, the bound in the Theorem 1.1 is reduced to $O(n^4 \sigma \lg \sigma)$. Note that σ is bounded by the input size plus the query size and so it is linear in the input size under combined complexity.[4] The maximum number of ground IDB atoms is $\tilde{O}(|D|^k)$ where k is the maximum arity of IDB atoms, where $\tilde{O}$ hides less significant factors such as p, λ and σ. Thus, Theorem 1.1 gives a polynomial bound on the convergence rate (in data complexity), and furthermore, as the bound depends on several parameters of the instances, it is more sensitive to properties of the data instance.

Related Works. Finding a least fixed point solution to a Datalog$^\circ$ program is equivalent to finding a least fixed point solution to a system of polynomial equations over a semiring. (Section 2 will explain why this is the case.) There is a large body of research on fixed points of multi-variate polynomial functions over semirings, which were studied by many communities since the 1960s. (For more details, see e.g. [5, 11, 13, 19].) In some special cases, such as closed semirings or ω-continuous semirings, there are non-iterative methods to find the fixpoint [5, 14]. In general, however, the iterative algorithm is still the most general.

One particularly interesting connection to our problem is Parikh's theorem [16] in formal language theory, discovered in 1960s. The theorem essentially states that, if we ignore the order of letters, then every context free language is isomorphic to a regular language. (See Section 2 for a formal statement.) Pilling [17] observed that the theorem can be generalized to be a statement about the least fixed point solution of a system of "regular equations". Hopkins and Kozen [9] generralized Pilling's theorem further to the least fixed point solution of a system of polynomial equations over a semiring.

The two recent papers in the literature that we directly build on are [11] and [10]. In [11] it is shown that the naïve evalution of Datalog$^\circ$ programs converges in $O((p+2)^n)$ steps; this bound is obtained by showing how to bound the convergence time for a high dimensional function in terms of the convergence time for a 1-dimensional function.

The paper [10] considers *linear* Datalog$^\circ$ programs, where the grounded *immediate consequence operator (ICO) is a linear function $\boldsymbol{f} : S^n \to S^n$ associated with a semiring $\mathbf{S} = (S, \oplus, \otimes, 0, 1)$ that can*

[4]Note that in [11] the parameter σ denoted the number of references to semiring elements, not the number of semiring elements referenced.

be expressed as $\boldsymbol{f}(x) = A \otimes x \oplus b$ where A is an n by n matrix with entries from the semiring. Then $f_i^{(q)}(0)$ becomes $\bigoplus_{W \in \mathcal{W}_q^i} Z(W)$, where $\mathcal{W}_q^i$ is the collection of all walks starting from ground atom i with at most q hops in the natural complete digraph underlying A, and $Z(W)$ is the product of a_{uv} for every edge $uv \in W$. The crux of the $O(pn^3)$ upper bound analysis in [10] was then that any walk W longer than $O(pn^3)$ must contain a cycle C where all the edges are traversed many times. The fact that adding $Z(W)$ didn't change the sum followed from the stability of the semiring element that is the product of the semiring values on the edges in C. Our analysis for the nonlinear case is more involved than the analysis for the linear cases because finding the collection of semiring values that will serve the role of the cycle C in the linear case is more involved. It is interesting to note that, however, the gap between the two cases is $O(n\sigma \lg \sigma)$.

Paper Organization. Section 2 covers background knowledge required to understand this result. Section 3 gives a brief technical overview of the proof of Theorem 1.1. Sections 4, 5, and 6 give the proof details. Finally Section 7 concludes the paper.

2 Background

2.1 Semirings

A *semiring* is a tuple $\boldsymbol{S} = (S, \oplus, \otimes, 0, 1)$ where $\oplus$ and $\otimes$ are binary operators on S, $(S, \oplus, 0)$ is a commutative monoid (meaning $\oplus$ is commutative and associative, and 0 is the identity for $\oplus$), $(S, \otimes, 1)$ is a monoid (meaning $\otimes$ is associative, and 1 is the identity for $\otimes$), $a \otimes 0 = 0 \otimes a = 0$ for every $a \in S$, and $\otimes$ distributes over $\oplus$. $\boldsymbol{S}$ is said to be *commutative* if $\otimes$ is commutative. Define

$$u^{(p)} := 1 \oplus u \oplus u^2 \oplus \cdots \oplus u^p,$$

where $u^i := u \otimes u \otimes \cdots \otimes u$ (i times). An element $u \in S$ is *p-stable* if $u^{(p)} = u^{(p+1)}$, and a semiring $\boldsymbol{S}$ is *p-stable* if every element $u \in S$ is p-stable.

A function $f : S^n \to S^n$ is p-stable if $f^{(p+1)}(\mathbf{0}) = f^{(p)}(\mathbf{0})$, where $\mathbf{0}$ is the all zero vector, and $f^{(k)}$ is the k-fold composition of f with itself. The *stability index* of f is the smallest p such that f is p-stable. See [7] for more background on semirings and stability.

Commonly used semirings include the Boolean semiring $(\{\mathsf{true}, \mathsf{false}\}, \vee, \wedge, \mathsf{false}, \mathsf{true})$, the (min-plus) tropical semiring $\mathsf{Trop}^+ = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$, and sum-product semirings such as $(\mathbb{R}, +, \times, 0, 1)$ and $(\mathbb{N}, +, \times, 0, 1)$.

The sum-product semirings above are not stable. Both the Boolean and the min-plus semirings are 0-stable, because

$$\begin{aligned} \mathsf{true} \vee x &= \mathsf{true} && \text{for any } x \in \{\mathsf{true}, \mathsf{false}\}, \\ \min\{0, x\} &= 0 && \text{for any } x \in \mathbb{R}_+ \cup \{\infty\}. \end{aligned}$$

Given a positive integer p, the tropical semiring $\mathsf{Trop}_p^+ = (\mathcal{B}_{p+1}, \oplus_p, \otimes_p, 0_p, 1_p)$ is p-stable, where the domain $\mathcal{B}_{p+1}$ are *bags* of $p+1$ values in $\mathbb{R}_+ \cup \{\infty\}$. The additive operator $x \oplus_p y$ returns the smallest $p+1$ elements from the bag-union of the bag x and y, and the multiplicative operator $x \otimes_p y$ returns the smallest $p+1$ elements of the bag $\{x_i + y_j \mid x_i \in x, y_j \in y\}$. Obviously, $\mathsf{Trop}_0^+ = \mathsf{Trop}^+$.

2.2 Datalog

A (traditional) Datalog [1] program P consists of a set of rules of the form:

$$R_0(\boldsymbol{X}_0) \text{ :- } R_1(\boldsymbol{X}_1) \wedge \cdots \wedge R_m(\boldsymbol{X}_m) \tag{4}$$

where $R_0, \ldots, R_m$ are predicate names (not necessarily distinct) and each $\boldsymbol{X}_i$ is a tuple of variables and/or constants. The atom $R_0(\boldsymbol{X}_0)$ is called the head, and the conjunction $R_1(\boldsymbol{X}_1) \wedge \cdots \wedge R_m(\boldsymbol{X}_m)$ is called the body. Multiple rules with the same head are interpreted as a disjunction. A predicate

that occurs in the head of some rule in P is called an *intensional database predicate* (IDB), otherwise it is called an *extensional database predicate* (EDB). The EDBs form the input, and the IDBs represent the output computed by the Datalog program. The finite set of all constants occurring in an EDB is called the *active domain*, and denoted ADom. An atom $R(\boldsymbol{X})$ is called a *ground atom* if all its arguments are constants. There is an implicit existential quantifier over the body for all variables that appear in the body, but not in the head, where the domain of the existential quantifier is ADom. Thus, a Datalog program can also be viewed as a collection of unions of conjunctive queries (UCQs), one UCQ for each IDB.

Example 1. *A classic example [1] of a* Datalog *program is the transitive closure program*

$$T(X,Y) \text{ :- } E(X,Y)$$
$$T(X,Y) \text{ :- } T(X,Z) \wedge E(Z,Y)$$

Here E is an EDB predicate, representing the edge relation of a directed graph, T is an IDB predicate, and ADom is the vertex set. Written as a UCQ, where the quantifications are explicit, this program is:

$$T(X,Y) \text{ :- } E(X,Y) \vee \exists_Z \ (T(X,Z) \wedge E(Z,Y)) \tag{5}$$

A Datalog program P can be thought of as a function, called the *immediate consequence operator (ICO)*, mapping a subset of ground IDB atoms to a subset of ground IDB atoms. (The ground EDB atoms are inputs and thus remain constants.) In particular, the ICO adds a ground (IDB) atom $R(\boldsymbol{x})$ to the output if it can be logically inferred by the input ground atoms via the rules of P. The iterative evaluation of a Datalog program works in rounds/steps, where on each round the ICO is applied to the current state, starting from the empty state.

In Example 1, the ICO starts by setting $T = \emptyset$, namely no pair of vertices are reachble from one another. Then, in the first round, the rule $T(X,Y)$:- $E(X,Y)$ fires for every edge (X,Y) in the graph. In the second round, in addition to the edges, the rule $T(X,Y)$:- $T(X,Z) \wedge E(Z,Y)$ fires for every pair of vertices X, Y that are reachable via a path of length 1, and thus we add to T all pairs (X,Y) of vertices that are reachable via a path of length 2. This process repeats until no new pairs of vertices can be added to T. The number of iterations is thus at most the number of reachable pairs of vertices, which is the output size.

2.3 Datalogo

Like Datalog programs, a Datalog$^\circ$ program consists of a set of rules, where the UCQs are replaced by *sum-sum-product queries* over a commutative semiring $\boldsymbol{S} = (S, \oplus, \otimes, 0, 1)$, where $\vee$ is replaced with $\oplus$ and $\wedge$ with $\otimes$. Specifically, in a Datalog$^\circ$ program each rule has the form:

$$R_0(X_0) \text{ :- } \bigoplus R_1(\boldsymbol{X}_1) \otimes \cdots \otimes R_m(\boldsymbol{X}_m) \tag{6}$$

where sum is over the ADom of the variables not in X_0. Multiple rules with the same head are combined using the $\oplus$ operation, which is the analog of combining rules using $\vee$ in Datalog.

Furthermore, each ground EDB or IDB atom is associated with an element of the semiring $\boldsymbol{S}$, and the non-zero elements associated with ground EDB atoms are specified in the input. A fixed point solution to the Datalog$^\circ$ program associates a semiring element to ground IDB atoms. Just like in Datalog, we do not have to explicitly represent the zero-assigned ground IDB atoms: every ground atom not in the output are implicitly mapped to 0.

Example 2. *The* Datalog$^\circ$*-version of the* Datalog *program given in line* (5) *with $\vee$ replaced by $\oplus$, $\wedge$ replaced by $\otimes$ and $\exists_Z$ replaced by $\bigoplus_Z$ is*

$$T(X,Y) \text{ :- } E(X,Y) \oplus \bigoplus_Z T(X,Z) \otimes E(Z,Y), \tag{7}$$

Here E is an EDB predicate, and T is an IDB predicate, $\oplus$ is the semiring addition operation, $\otimes$ is the semiring multiplication operation and $\bigoplus_Z$ is aggregation, that is an iterative application of $\oplus$ over ADom.

When interpreted over the Boolean semiring, the Datalog$^\circ$ *program in* (7) *is the transitive closure program from Example 1.*

When interpreted over the tropical semiring $\mathsf{Trop}^+ = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$, *the* Datalog$^\circ$ *program in* (7) *solves the classic* All-Pairs-Shortest-Path *(APSP) problem, which computes the shortest path length $T(X, Y)$ between all pairs X, Y of vertices in a directed graph specified by an edge relation $E(X, Y)$, where the semiring element associated with $E(X, Y)$ is the length of the directed edge (X, Y).*

$$T(X, Y) \coloneq \min\left(E(X, Y), \min_Z(T(X, Z) + E(Z, Y))\right) \tag{8}$$

When interpreted over the sum-product semiring $(\mathbb{R}, +, \times, 0, 1)$, the Datalog$^\circ$ *program in* (7) *computes, for every pair (X, Y) of vertices, the sum of the products of the lengths of all paths from X to Y in the graph. This process generally does not converge for graphs with cycles, and it is an example of how an unstable semiring can lead to non-convergence.*

A Datalog$^\circ$ program can be thought of as an immediate consequence operator (ICO). A simple way to understand the semantics of Datalog$^\circ$ is to think of each body predicate R_i in (6) as a function from the domain of $\boldsymbol{X}_i$ to the domain S of the semiring. The functional value $R_i(\boldsymbol{c}_i)$ for a particular binding $\boldsymbol{c}_i$ in the domain of $\boldsymbol{X}_i$ is the value assigned to the ground atom $R_i(\boldsymbol{c}_i)$. The rule (6) is thus exactly a *sum-product query* (or a *functional aggregate query* [2] over one semiring), and multiple rules with the same head are combined into a *sum-sum-product* query. The Datalog$^\circ$ program containing these queries compute new (IDB) functions from old (IDB and EDB) functions, using the sums and products from the semiring.

As mentioned in the introduction, these EDBs and IDBs are called *K-relations* in the landmark paper [8]. While every EDB/IDB of arity ℓ needs up to $|\mathsf{ADom}|^\ell$ tuples to represent, we only need to explicitly materialize the non-zero-valued tuples, and use this number to denote the "size" of the representation.

The iterative evaluation of a Datalog$^\circ$ program works by initially assigning all IDB "functions" to be identically 0 (i.e. their ground atoms are assigned with 0). The ICO is then repeatedly applied to the current IDB state. In the context of the Datalog$^\circ$ program in (8), initially all $T(x, z)$ are assigned with $+\infty$ (the 0 of the tropical semiring), and the rule (8) effectively is the well-known Bellman-Ford algorithm [4]. The *convergence rate* of a Datalog$^\circ$ program is the stability index of its ICO.

A Datalog$^\circ$ program is linear if every rule (6) has no more than one IDB predicate in its body. The Datalog$^\circ$ program in Example 2 is linear. While many natural Datalog$^\circ$ programs are linear, there are also natural nonlinear Datalog$^\circ$ programs.

Example 3. *As a classic example of a nonlinear* Datalog$^\circ$ *program, consider the following alternate formulation of APSP, which is equivalent to* (8)

$$T(X, Y) \coloneq \min\left(E(X, Y), \min_Z(T(X, Z) + T(Z, Y))\right) \tag{9}$$

It is interesting to note that, for unit-edge lengths, this program takes a logarithmic number of iterations to converge, because at the ith iteration we will have discovered all shortest paths of length $\leq 2^i$. However, many iterations take longer to compute, because we are joining T with itself, and T may be much larger than E.

2.4 Grounding the ICO

Since the final associated semiring values of the ground IDB atoms are not initially known, it is natural to think of them as (IDB) variables. Then the grounded version of the ICO of a Datalog$^\circ$ program is a map $\boldsymbol{f} : S^n \to S^n$, where S is the semiring domain, and n is the number of ground IDB atoms that ever have a nonzero value at some point in the iterative evaluation of the program. For instance, in (8), there would be one variable $T(x, y)$ for each pair (x, y) of vertices where there is a directed path from x to y in the graph. So the grounded version of the ICO of a Datalog$^\circ$ program has the following form:

$$\begin{aligned} X_1 &:\text{-}\ f_1(X_1, \ldots, X_n) \\ &\ldots \\ X_n &:\text{-}\ f_n(X_1, \ldots, X_n) \end{aligned} \tag{10}$$

where the X_i's are the IDB variables, and f_i is the component of $\boldsymbol{f}$ corresponding to the IDB variable X_i. Note that each component function f_i is a multivariate polynomial in the IDB variables of degree at most the maximum number of factors in any product in the body of some rule (6) in the Datalog$^\circ$ program. After q iterations of the iterative evaluation of a Datalog$^\circ$ program, the semiring value associated with the ground atom corresponding to X_i will be:

$$f_i^{(q)}(\mathbf{0}) \tag{11}$$

Example 4. *Consider the binary recursive formulation in* (9)*, written over a generic semiring.*

$$T(X, Y) :\text{-}\ E(X, Y) \oplus \bigoplus_Z T(X, Z) \otimes T(Z, Y) \tag{12}$$

Suppose ADom $= \{1, 2, 3, 4\}$*, and the input EDB contains ground EDB atoms* $E(1, 2), E(2, 3), E(3, 4)$*, with corresponding (constant) semiring values* e_{12}, e_{23}, e_{34}*. Then there will be 16 equations and 16 variables in the grounded ICO; For each* $a, b \in \{1, \ldots, 4\}$ *there will a variable* X_{ab}*, and an equation of the form:*

$$X_{ab} :\text{-}\ e_{ab} \oplus \bigoplus_{i \in [4]} X_{ai} \otimes X_{ib}$$

But, as many of these variables will always be 0*, they are "inactive" and thus they can effectively be ignored from the grounded ICO formulation. Thus effectively one can think of the grounded ICO as having the following* 6 *variables and 6 equations:*

$$\begin{aligned} X_{12} &:\text{-}\ e_{12} & X_{23} &:\text{-}\ e_{23} \\ X_{13} &:\text{-}\ X_{12} \otimes X_{23} & X_{24} &:\text{-}\ X_{23} \otimes X_{34} \\ X_{14} &:\text{-}\ X_{12} \otimes X_{24} \oplus X_{13} \otimes X_{34} & X_{34} &:\text{-}\ e_{34}. \end{aligned}$$

2.5 Context Free Languages

It is convenient to reason about the formal expansion of $f_i^{(q)}(\mathbf{0})$ using context-free languages (CFL). See [12, 15] for an introduction to CFLs. As mentioned in the introduction, this is effectively the connection that Pillings [17] and Hopkins and Kozen [9] made between CFLs and the least fixed point of a system of polynomial equations over a semiring. To explain the connection, it is probably best to start with a concrete example.

Example 5. *The map* $\boldsymbol{f} : S^2 \to S^2$ *where*

$$\begin{aligned} f_1(A, B) &= (a \otimes A \otimes B) \oplus (b \otimes B) \oplus c \\ f_2(A, B) &= (c \otimes A \otimes B) \oplus (b \otimes A) \oplus a \end{aligned}$$

can be represented by the following context free grammar G:

$$A \longrightarrow aAB \mid bB \mid c \qquad\qquad B \longrightarrow cAB \mid bA \mid a$$

Suppose we would like apply the map $\boldsymbol{f}$ twice to write down an expression for

$$\boldsymbol{f}^{(2)}(A,B) = \boldsymbol{f}\left(\boldsymbol{f}\left(\begin{bmatrix}A\\B\end{bmatrix}\right)\right) = \boldsymbol{f}\left(\begin{bmatrix}(a\otimes A\otimes B)\oplus(b\otimes B)\oplus c\\(c\otimes A\otimes B)\oplus(b\otimes A)\oplus a\end{bmatrix}\right) = \cdots$$

Then, this can be done by expanding the context free grammar G to depth 2, where the first component of $\boldsymbol{f}^{(2)}(A,B)$ is the sum of the products of the leaves of the parse trees of depth at most 2 and rooted at A, and the second component corresponds to the parse trees rooted at B.

More generally the variables (e.g. A, B) in f become non-terminals in G, the constants (e.g. a, b, c) in f become the terminals in G, multiplication $\otimes$ in f becomes concatenation in G, and addition $\oplus$ in f becomes the or operator | in G. Given a parse tree T for the grammar, define the *yield* $Y(T)$ of T to be the string of terminal symbols at the leaves of T, and the *product yield* $Z(T)$ to be the product of the semiring values in $Y(T)$. Let $\mathcal{T}_q^i$ denote the set of all parse trees with starting non-terminal X_i, and depth $\leq q$. Note that then:

$$f_i^{(q)}(0) = \bigoplus\nolimits_{T\in\mathcal{T}_q^i} Z(T). \tag{13}$$

That is, the value of the semiring value associated with a IDB variable X_i after q iterations is the sum of the product of the leaves of parse trees of depth at most q and rooted at X_i.

2.6 Parikh's Theorem

Since our semiring is commutative, many summands in the sum (13) are identical. Studying CFLs with commutative alphabet was exactly the goal of Parikh's theorem [16]. The upper bound on the time to convergence for iterative evaluation of Datalog° programs in [11] essentially relied on black-box application of Parikh's theorem. (See standard computability textbooks such as [12] for an introduction to Parikh's theorem.) The *Parikh image* of a word $w \in \Sigma^*$, denoted by $\Psi(w)$, is the vector $\Psi(w) = (k_1, \ldots, k_\sigma) \in \mathbb{N}^\sigma$ where k_i is the number of occurrences of the letter $a_i \in \Sigma$ that occur in the word w (So $|\Sigma| = \sigma$). Similarly, for a language L, define $\Psi(L) := \{\Psi(w) \mid w \in L\}$. Then using our assumption that the underlying semiring $\boldsymbol{S}$ is commutative, we observe that

$$f_i^{(q)}(\mathbf{0}) = \bigoplus\nolimits_{T\in\mathcal{T}_q^i} Z(T) = \bigoplus\nolimits_{T\in\mathcal{T}_q^i} \bigotimes\nolimits_{j=1}^{\sigma} a_j^{\Psi_j(Y(T))} \tag{14}$$

where $\Psi_j(w)$ is component j of the Parikh image. One version of Parikh's theorem [16] states that the Parikh images of the words in a context free language forms a semi-linear set. A set is *semi-linear* if it is a finite union of linear sets. A set $\mathcal{L} \subseteq \mathbb{N}^\sigma$ is said to be *linear* if there exist offset vector $\boldsymbol{v}_0$ and basis vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_\ell \in \mathbb{N}^\sigma$ such that $\mathcal{L}$ is the span of these vectors, that is if:

$$\mathcal{L} = \{\boldsymbol{v}_0 + k_1\boldsymbol{v}_1 + \ldots + k_\ell\boldsymbol{v}_\ell \mid k_1, \ldots, k_\ell \in \mathbb{N}\}.$$

If a vector

$$\boldsymbol{v} = \boldsymbol{v}_0 + k_1\boldsymbol{v}_1 + \ldots + k_\ell\boldsymbol{v}_\ell$$

then we say $(k_1, \ldots, k_\ell)$ is a linear representation of v within $\mathcal{L}$.

The textbook proof of Parikh's theorem (see [12]) uses what we will call a wedge. A wedge within a parse tree T can be specified by identifying two internal nodes in the parse tree that correspond to the same nonterminal, say A, and that have an ancestor-descendent relation. The corresponding wedge W then consists of the nodes in the parse tree that are descendents of the top A, but not the bottom A. See Figure 1.

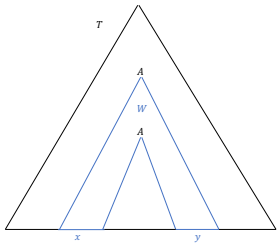

Fig. 1. Illustration of a wedge W

3 Technical Overview

We now give a technical overview of the proof of Theorem 1.1. To build intuition, let us first consider the analysis of linear Datalog° programs from [10], where the critical insight was that every walk of length $\Omega(pn^3)$ must contain a cycle C in which all edges are traversed p times. For general Datalog° programs, the natural way to extend the analysis of the linear case is to identify a type of substructure of a parse tree that has similar properties to the cycle used in the analysis of linear Datalog° programs. So in particular, the removal of such a substructure will result in a valid parse tree, and any sufficiently deep parse tree must have many disjoint copies of a particular substructure of this type. Given knowledge of the standard proof of Parikh's theorem [12], the first natural candidate substructure type is a wedge. It is relatively straight-forward to observe that if a parse tree is sufficiently deep, it must contain p disjoint copies of some wedge W. The proof that adding/removing such a wedge does not change the sum then follows from the p-stability of the terminals in W. However, it is not difficult to see that one can not improve the bound on the depth to subexponential using wedges as the type of substructure as there are parse trees that are exponentially deep and that do not contain repeated wedges.

Thus we turn to the semilinear representation of the context free language that is guaranteed to exist by Parikh's theorem. First we need to better understand the relationship between a linear set and the parse trees for strings in this linear set than is given by Parikh's theorem. We first show, in Theorem 3.1, that there is a particular semilinear representation $\mathcal{M}$ in which every linear set consists of offset and basis vectors with small 1-norm. Theorem 3.1 also shows that one can essentially polynomially bound the 1-norm of the linear representation of the yield of a parse tree by the depth of the parse tree, and conversely polynomially bound the depth of a valid parse tree by the 1-norm of a linear representation. Thus, it would be sufficient to show that if the yield of a deep parse tree has a linear representation with small support then this parse tree does not add to the sum; This is because by the pigeon hole principle it would have to be the case that the coefficient of some basis vector in the linear representation would have to be large, and thus one could appeal to the p-stability of that basis vector. But this is not a general proof as the number of basis vectors in a linear set may be exponential. Thus to make this proof technique general, we need to show that for every word in the context free language there is a linear set where the yield of that word has a linear representation will small support. This is accomplished in Lemma 3.2, the proof of which uses the p-stability of the underlying semiring.

Theorem 3.1. *Let L be a context free language generated by a grammar $G = (N, \Sigma, R, S)$, where N is the collection of nonterminals, Σ is the collection of terminals, R is the collection of rules, and $S \in N$ is the start non-terminal. Let n be the cardinality of N and let λ be the maximum number of symbols on the righthand side of any rule. Then there exists a finite semi-linear set $\mathcal{M}$ with the following properties:*

(1) Every linear set $\mathcal{L} \in \mathcal{M}$ has an associated offset vector v_0 and basis vectors $v_1, \ldots v_\ell$ whose 1-norm is at most $\lambda^{n(n+3)/2}$.

(2) $\mathcal{M} = \Psi(L)$.

(a) $\mathcal{M} \subseteq \Psi(L)$. Further, for each vector $\boldsymbol{v} = \boldsymbol{v}_0 + k_1\boldsymbol{v}_1 + \ldots + k_\ell\boldsymbol{v}_\ell$ in the span of some $\mathcal{L} \in \mathcal{M}$, there is a word $w \in L$, with $\Psi(w) = \boldsymbol{v}$, where w can be generated by a parse tree with depth at most $(k+1)n(n+3)/2$, where $k = k_1 + k_2 + \ldots + k_\ell$.

(b) $\mathcal{M} \supseteq \Psi(L)$. Further, for any parse tree T of depth d such that $Y(T) \in L$, there exists a linear representation of $\Psi(Y(T)) = \boldsymbol{v}_0 + k_1\boldsymbol{v}_1 + \ldots + k_\ell\boldsymbol{v}_\ell$ within some $\mathcal{L} \in \mathcal{M}$ such that $k + 1 \geq d/(n(n+3)/2)$, where $k = k_1 + k_2 + \ldots + k_\ell$.

The proof of Theorem 3.1 is given in Section 4. The most important way that Theorem 3.1 extends the standard version of Parikh's theorem is property (2)(a), which upper bounds the depth of some parse tree of a word by the 1-norm of the representation of that word. The bound given in the textbook proof [12] of Parikh's theorem gives a depth bound that is exponentially large. To achieve property (2)(a), our proof contains a constructive forward process $\mathcal{P}$ that creates a linear set $\mathcal{L} \in \mathcal{M}$ from the parse tree T of some word $w \in L$ by removing wedges from T. We are careful to design $\mathcal{P}$ so that it is reversible; that is, to recover a parse tree from a linear representation we can just reverse the process $\mathcal{P}$. To accomplish this we need that in the forward process every wedge that is removed does not remove any nonterminal from the parse tree. Our process $\mathcal{P}$ ensures that the parse tree for the offset and the wedges for the basis vectors have depth $O(n^2)$.

Property (2)(b) establishes the converse and further states that if a parse tree has a large depth, then there must exist a linear representation of its Parikh's image within some $\mathcal{L} \in \mathcal{M}$ that has a large 1-norm. The proof of this property is similar to that of Property (2)(a).

Lemma 3.2. *Let L be an arbitrary context free language. Let $h = 2(\sigma(n(n+3)/2)\lg(\lambda+1) + 4\sigma\lg\sigma)$. Let $\mathcal{M}$ be the semilinear set that is guaranteed to exist in Theorem 3.1. Let w be a word in L. Let $\mathcal{L}$ in $\mathcal{M}$ be a linear set such that $\Psi(w) \in \mathcal{L}$. Let $\Psi(w) = \boldsymbol{v}_0 + k_1\boldsymbol{v}_1 + \ldots + k_m\boldsymbol{v}_m$ be a linear representation of $\Psi(w)$ with respect to the offset and basis vectors of $\mathcal{L}$. Then there exists another linear representation $\Psi(w) = \boldsymbol{v}_0 + k'_1\boldsymbol{v}'_1 + \ldots + k'_h\boldsymbol{v}'_h$ of $\Psi(w)$ with respect to h basis vectors of $\mathcal{L}$ such that $\sum_{i=1}^{m} k_i \leq \sum_{i=1}^{h} k'_i$.*

The proof of Lemma 3.2, given in Section 5, uses properties of $\mathcal{L}$ established in our strengthened version of Parikh's theorem (Theorem 3.1), and the pigeon hole principle to establish that any word in $\mathcal{L}$ that has a linear representation with respect to $\mathcal{L}$ with large support, also has a linear representation with smaller support. Here, we do not decrease the 1-norm of the linear representation with respect to $\mathcal{L}$, which will be important for the proof of our main result. In particular we use the finding that all the offset and basis vectors have (relatively) small 1-norms. This makes formal the intuition that one might draw from standard vector spaces that the number of basis vectors needed to represent a vector/word in the span of some basis vectors shouldn't be more than the dimensionality of the spanned space.

Finally in Section 6 we use Lemma 3.2 to prove Theorem 1.1. To show that for sufficiently large q it is the case that $f_i^{(q)}(\mathbf{0}) = f_i^{(q+1)}(\mathbf{0})$, we show that for every tree $T' \in \mathcal{T}_{q+1}^i \setminus \mathcal{T}_q^i$, the corresponding summand $Z(T')$ is "absorbed" by the earlier terms, that is:

$$\bigoplus_{T \in \mathcal{T}_q^i} Z(T) \oplus Z(T') = \bigoplus_{T \in \mathcal{T}_q^i} Z(T) \tag{15}$$

From property (2)(b) of Theorem 3.1 we know that that there exists a linear representation of $\Psi(Y(T'))$ that has a large 1-norm; and from Lemma 3.2 we know that there is a linear representation with the same or larger 1-norm of $\Psi(Y(T'))$ with small support. Thus we can conclude by the pigeonhole principle that one of the basis vectors, in this small support linear representation, must

have a coefficient greater than the stability p of the ground set. Finally, Eqn. (15) follows from the stability of the semiring element that corresponds to the semiring element that is the "product" of that basis vector.

4 The Strengthened Parikh's Theorem

Our goal in this section is to prove Theorem 3.1. cummutitave semirings, we will ignore orderings of terminals in a string. Thus, for simplicity, we may allow a string to denote the product of terminals in it. In the analysis it will be convenient to define the depth of strings.

We begin our proof by defining a particular semi-linear set $\mathcal{M}$ with the desired properties. Let $c := n(n+3)/2$ throughout this section; recall that n denotes the number of non-terminals. Let $\mathcal{T}_c^S$ denote the set of all parse trees (starting with the non-terminal S) of depth at most c. Recall that T's yield, denoted as $Y(T)$, is the word obtained by a parse tree T. For a wedge W, $Y(W)$ is analogously defined by ignoring the unique non-terminal leaf node in the wedge W. Let $N(T)$ denote the set of non-terminals that appear in T. For a parse tree T and a non-terminal $A \in N(T)$, let $\mathcal{W}_c^A(T)$ be the collection of wedges W where the root of W is A, $N(W) \subseteq N(T)$, and W has height/depth at most c (Recall that the depth/height of a tree is the number of hops in the longest simple root to leaf path). Note that the definition of $\mathcal{W}_c^A(T)$ depends only on the non-terminals in T, and not on the structure of T; so in particular there is no requirement that a wedge $W \in \mathcal{W}_c^A(T)$ be a subtree of T. Let $\mathcal{B}_c(A,T) := \{Y(W) \mid W \in \mathcal{W}_c^A(T)\}$. In other words, starting with A, $\mathcal{B}_c(A)$ is the collection of all words (more precisely the product of the terminals in each word) that one can obtain by a parse tree rooted at A of depth at most c where all leaf nodes are terminals, except one leaf being A. Notably, while we use notations $\mathcal{W}_c^A(T)$ and $\mathcal{B}_c(A,T)$ for notational brevity, their dependence is on $N(T)$ rather than T. For notational brevity, we may use $\mathcal{B}(A,T)$ instead of $\mathcal{B}_c(A,T)$.

Then, for each tree T in $\mathcal{T}_c^S$, we define a linear set where the offset vector is $\Psi(Y(T))$ and the basis vectors are $\cup_{V \in N(T)} \Psi(\mathcal{B}(V,T))$. Here, $\Psi(L')$ denotes the collection of vectors corresponding to a subset of words, L'. Notice that because of the way we created the offset vector and basis vectors, there is a parse tree in $\mathcal{T}_c^S$ corresponding to each offset vector and a wedge corresponding to each basis vector, all of depth at most c.

In the following we recall the definition of wedges (Figure 1) and define how to index them. For an arbitrary parse tree T we will map it to a tree in $\mathcal{T}_c^S$ by iteratively removing a wedge.

Definition 4.1. Define a *wedge* of a parse tree T as follows. Consider two occurrences of a non-terminal A in T where one is an ancestor of the other. Let A' be the ancestor node and A'' the descendant node. The wedge induced by the pair (A', A'') is defined as the subtree of T rooted at A' with the subtree rooted at A'' removed. The wedge is denoted as $W_T(A', A'')$. The wedge's depth is defined as the maximum number of edges from A' to a leaf node in $W_T(A', A'')$.

Lemma 4.2 states the properties that we will want to maintain during each step of our iterative process to decompose our parse tree T.

Lemma 4.2. *Given a parse tree T of depth at least c starting with non-terminal S, we can obtain a parse tree T' starting with S that satisfies the following:*

(1) (Preserving Non-terminals) $N(T) = N(T')$.
(2) (Reversibility) T can be obtained by replacing one non-terminal A in T' with a wedge $W \in \mathcal{W}_c^A(T')$ for some $A \in N(T')$.

Alternatively, the second property means that T can be obtained from T' by augmenting T' with a wedge W of depth at most c corresponding to a vector in $\Psi(\mathcal{B}(A,T'))$ for some non-terminal A in $N(T')$. Here it is worth noting that $\mathcal{W}_c^A(T) = \mathcal{W}_c^A(T')$ because $N(T) = N(T')$. See Figure 2 for an illustration of reversibility.

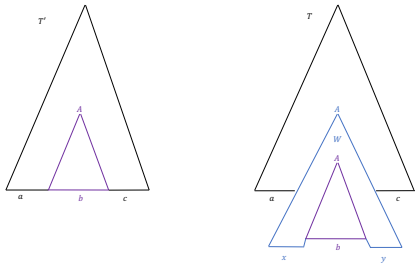

Fig. 2. The right tree T is recovered from the left tree T' by augmenting the wedge W.

The first property in the lemma is worth special attention. Suppose we obtained T' from T by repeatedly applying the lemma, but without guaranteeing the first property. Suppose $\{W_1, W_2, \dots\}$ are the wedges we removed in the process; so W_i is the wedge removed in iteration i. Then, we may not be able to augment T' with an arbitrary subset of the wedges, which is critical to establish the $\Psi(L) \supseteq \mathcal{M}$ direction of the first property of Theorem 3.1.

To show Lemma 4.2, consider an arbitrary parse tree T of depth more than c. We show how to obtain T' by collapsing a wedge induced by two occurrences of the same non-terminal. Below, we describe how we find a "good" pair of two occurrences of the same non-terminal we want to collapse. We first define what makes pairs good in the following.

Definition 4.3. For a given parse tree T, we say a pair of ascendant and descendant nodes (A', A'') of the same non-terminal A is good if it satisfies the following:

- Let T' be the tree T with the wedge $W_T(A', A'')$ removed. We have $N(T) = N(T')$.
- The height of A' is at most c in T. In other words, the subtree of T rooted at A' has depth at most c.

In the following we will show that a tree of large depth must have a good pair. Note that we will immediately have Lemma 4.2 as corollary if we prove the following lemma.

Lemma 4.4. *A parse tree T of depth at least c has a good pair of nodes.*

Proof. We prove the lemma by an induction on the number of non-terminals, n. Let $c_n := (n+1) + n + \dots + 2$; notably, $c_n = c$. Then, the second property of Definition 4.3 becomes that the height of A' is at most $c_{n'}$ when T has at most n' non-terminals. Consider an arbitrary node v of the largest depth, which must be at least c_n. Since the base case $n = 1$ is trivial, suppose $n \geq 2$. Consider the unique path from v to the root non-terminal S in T. The height of a node u on the path is defined as the number of nodes below u on the path, including u.

Consider walking from u towards the root. On this path, consider the first time two occurrences of the *same* non-terminal V_1 appear. Say they appear at nodes $v(V_1)$ and $u(V_1)$, where $u(V_1)$ is an ascendant of $v(V_1)$. Observe that $u(V_1)$ has height at most $n+1$ due to the pigeon hole principle. This is because some non-terminal must repeat among $n+1$ nodes.

If $(u(V_1), v(V_1))$ is a good pair, we are done. If not, it means that the wedge $W_T(u(V_1), v(V_1))$ must include a non-terminal that *doesn't appear anywhere else in the tree.* Consider the tree T_2 with the subtree rooted at $u(V_1)$ removed. This subtree T_2 has at most $n-1$ non-terminals and has depth at least $c_{n-1} = c_n - (n+1)$. By induction, this implies that T_2 must have a good pair (u_2, v_2).

We verity that the pair (u_2, v_2) remains to be good with respect to T as well: First, u_2 has height at most $(n+1) + (n + (n-1) + \dots + 2) = c$ in the tree T. Second, $W_T(u_2, v_2)$ does not intersect the subtree rooted at $u(V_1)$, and therefore, the set of non-terminals remains unchanged after removing the wedge from T, just as it does when we remove the wedge from T_2. □

We are now ready to prove Theorem 3.1.

Property (1). By definition of the offset and basis vectors, it immediately follows that their depth is at most c. Furthermore, the 1-norm of any of them is at most λ^c because each node has at most λ children.

Property (2)(b). Given a parse tree T for $w \in L$, suppose we obtained a sequence of trees $T_0 = T, T_1, \ldots, T_\eta$ by repeatedly applying Lemma 4.2, where $T_\eta \in \mathcal{T}_c^S$ and T_i is obtained from T_{i-1} by deleting a wedge W_i in $\mathcal{W}_c^A(T)$ for some non-terminal A in $N(T) = N(T_\eta)$; note $\mathcal{W}_c^A(T) = \mathcal{W}_c^A(T_\eta)$ since $N(T) = N(T_1) = \ldots = N(T_\eta)$. Let $\boldsymbol{b}_i = \Psi(Y(W_i))$. Clearly, $\Psi(w)$ can be expressed as $\Psi(Y(T_\eta)) + \sum_{i=1}^{\eta} \boldsymbol{b}_i$.

Since we created a linear set for each parse tree in $\mathcal{T}_c^S$, thus for T_η, this is a linear representation within $\mathcal{L}$ which consists of offset vector $\Psi(Y(T_\eta))$ and basis vectors $\cup_{V \in N(T_\eta)} \Psi(\mathcal{B}(V, T_\eta))$. This proves $\mathcal{M} \supseteq \Psi(L)$.

We now show that the 1-norm of $\Psi(w)$'s linear representation, $\eta + 1$, is large. Suppose that the parse tree T has depth d. The two trees T_{i-1} and T_i have depths differing by at most c since we obtained T_i from T_{i-1} by deleting a wedge of depth at most c. Further, the last tree T_η has depth at most c as well. Thus, $c(\eta + 1) \geq d$. Since $\eta = k$ where k is described as in the theorem, we have proven this property.

Property (2)(a). Conversely, suppose w has a linear representation within some $\mathcal{L} \in \mathcal{M}$. Say the linear representation is $\Psi(Y(T')) + \sum_{i=1}^{k} \boldsymbol{b}_i$ for some $T' \in \mathcal{T}_c^S$. Note that for each basis vector $\boldsymbol{b}_i$, there exists $V_i \in N(T')$ such that $\boldsymbol{b}_i \in \Psi(\mathcal{B}(V_i, T'))$. Because of the way we defined linear sets, $\boldsymbol{b}_i = \Psi(Y(W_i))$ for some $W_i \in \mathcal{W}_c^{V_i}(T')$. We can augment T' with the wedges W_i in an arbitrary order. Adding each wedge W_i increases the tree depth by at most c. By repeating this for each $\boldsymbol{b}_i$, we obtain a parse tree T such that $\Psi(Y(T)) = \Psi(w)$. This proves $\mathcal{M} \subseteq \Psi(L)$ and T has depth at most at most $c(k + 1)$.

Together, the proof of the above properties prove Theorem 3.1.

5 Small Support Representations

In this section we prove Lemma 3.2. The lemma shows that any linear representation of the Parikh image of a word in a context-free language L can be converted into another linear representation with no smaller 1-norm, but with small support.

Lemma 5.1. *[Lemma 3.2 Restated] Let L be an arbitrary context free language. Let $h := 2(\sigma(n(n+3)/2)\lg(\lambda+1) + 4\sigma\lg\sigma)$. Let $\mathcal{M}$ be the semilinear set that is guaranteed to exist in Theorem 3.1. Let w be a word in L. Let $\mathcal{L}$ in $\mathcal{M}$ be a linear set such that $\Psi(w) \in \mathcal{L}$. Let $\Psi(w) = \boldsymbol{v}_0 + k_1\boldsymbol{v}_1 + \ldots + k_m\boldsymbol{v}_m$ be a linear representation of $\Psi(w)$ with respect to the offset and basis vectors of $\mathcal{L}$. Then there exists another linear representation $\Psi(w) = \boldsymbol{v}_0 + k_1'\boldsymbol{v}_1' + \ldots + k_h'\boldsymbol{v}_h'$ of $\Psi(w)$ with respect to h basis vectors of $\mathcal{L}$ such that $\sum_{i=1}^{m} k_i \leq \sum_{i=1}^{h} k_i'$.*

Proof. To streamline our analysis, we will assume that $\lambda \geq 2$. This is without loss of generality because in the case that $\lambda = 1$ we can add an unused terminal to Σ. The value of λ will increase by one in the final bound for this boundary case. For an arbitrary word w in our language L, suppose we are given a linear representation of $\Psi(w)$,

$$\boldsymbol{v}_0 + k_1\boldsymbol{v}_1 + \ldots + k_m\boldsymbol{v}_m,$$

where

$$k_1, k_2, \ldots k_m > 0 \text{ and}$$

$$m > h := 2(\sigma(n(n+3)/2)\lg\lambda + 4\sigma\lg\sigma).$$

Without loss of generality, we can assume that for any $i \in \{0, 1, 2, \ldots, m\}$, $||\boldsymbol{v}_i||_1 \leq M := \lambda^{n(n+3)/2}$ thanks to Theorem 3.1 (1).

To prove the lemma, it suffices to find another linear representation of $\Psi(w)$,

$$\boldsymbol{v}_0 + k_1'\boldsymbol{v}_1 + \ldots + k_m'\boldsymbol{v}_m \tag{16}$$

such that

$$k_1 + k_2 + \ldots + k_m \leq k_1' + k_2' + \ldots + k_m' \text{ and} \tag{17}$$

$$k_i' = 0 \text{ for some } i \in [m] := \{1, 2, \ldots, m\}. \tag{18}$$

A key step to our proof is showing that there exist two distinct subsets H_1 and H_2 of $[m]$ such that

$$\sum_{i \in H_1} \boldsymbol{v}_i = \sum_{i \in H_2} \boldsymbol{v}_i \tag{19}$$

We will prove this claim by proving that

$$K := |\{\sum_{i \in H} \boldsymbol{v}_i \mid H \subseteq [m]\}| < 2^m. \tag{20}$$

Note that K is the number of distinct vectors we can generate by summing a subset of vectors from $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$. The existence of the desired pair of H_1 and H_2 satisfying (19) will then follow from pigeonhole principle.

Since $||\boldsymbol{v}_i||_1 \leq M$ for all $i \in [m]$, we have $||\sum_{i \in H} \boldsymbol{v}_i||_1 \leq Mm$ for any $H \subseteq [m]$. We use the following well-known fact: the number of distinct vectors in $\mathbb{N}^d$ with 1-norm of k is exactly $\binom{k+d-1}{d-1} \leq (k+d-1)^{d-1}$ (see [20]). In our case, $k \leq Mm$ and $d = \sigma$. Thus, we have $K \leq (Mm + \sigma - 1)^{\sigma-1}(Mm+1) \leq (Mm+\sigma)^\sigma$. If $\sigma = 1$ (there exists only one terminal), we have a tighter bound of $K \leq M + (M-1) + \ldots + M - (m-1) = m(2M - (m-1))/2$.

To prove (20), it remains to show that $(Mm+\sigma)^\sigma < 2^m$ when $\sigma \geq 2$ and that $m(2M-m+1)/2 < 2^m$ when $\sigma = 1$. We consider two cases: $\sigma \geq 2$ and $\sigma = 1$.

Case i: $\sigma \geq 2$**.** We shall now establish

$$2^m > (Mm+\sigma)^\sigma \text{ when } \sigma \geq 2 \tag{21}$$

By taking the logarithm, the inequality (21) is equivalent to:

$$m > \sigma \lg(Mm + \sigma) \tag{22}$$

Since $M \leq \lambda^{n(n+3)/2}$, the following equation would imply Eqn. (22):

$$m > \sigma \lg(m\lambda^{n(n+3)/2} + \sigma) \tag{23}$$

Using the fact that $\lg x$ is sub-additive when $x \geq 2$ and the assumptions that $\lambda, \sigma \geq 2$, we have

$$\sigma \lg(m\lambda^{n(n+3)/2}) + \sigma \lg \sigma \geq \sigma \lg(m\lambda^{n(n+3)/2} + \sigma) \tag{24}$$

Thus, it is sufficient to show:

$$\begin{aligned} m &> \sigma \lg(m\lambda^{n(n+3)/2}) + \sigma \lg \sigma \\ &= \sigma \lg m + \sigma(n(n+3)/2) \lg \lambda + \sigma \lg \sigma \\ \Leftrightarrow\ m - \sigma \lg m &> \sigma(n(n+3)/2) \lg \lambda + \sigma \lg \sigma \end{aligned} \tag{25}$$

We show that

$$m - \sigma \lg m \geq m/2$$

when $m \geq 8\sigma \lg \sigma$: Since $m/2 - \sigma \lg m$ is increasing in m when $m \geq 8\sigma \lg \sigma$, we have $m/2 - \sigma \lg m \geq 4\sigma \lg \sigma - \sigma \lg(8\sigma \lg \sigma) = \sigma \lg(\sigma^3/(8 \lg \sigma)) \geq 0$ when $\sigma \geq 2$, as desired.

Thus, we have

$$m - \sigma \lg m \geq m/2 > \sigma(n(n+3)/2) \lg \lambda + 4\sigma \lg \sigma, \tag{26}$$

where the second inequality follows from the fact that $m > h$. From Eqn. (21), (22), (23), (24), (25), and (26) we have $2^m > K$ when $\sigma \geq 2$.

Case ii: $\sigma = 1$**.** If $\sigma = 1$, as mentioned above, we have

$$\begin{aligned} K &\leq M + (M-1) + \ldots + M - (m-1) \\ &= m(2M - (m-1))/2 \\ &< Mm \\ &\leq M2^{m/2} \quad [\text{Since } m \geq 2] \\ &\leq \lambda^{n(n+3)/2} 2^{m/2} \\ &\leq 2^m. \end{aligned}$$

The last inequality is true due to the assumption that $\sigma = 1$ and

$$m > h = 2(\sigma(n(n+3)/2) \lg \lambda + 4\sigma \lg \sigma) \geq n(n+3) \lg \lambda$$

Thus, we have shown that $2^m > K$ for all $\sigma \geq 1$, which establishes the existence of $H_1 \neq H_2 \subseteq [m]$ satisfying Eqn. (19).

We now explain how to construct a new representation of $\Psi(w)$ that contains less basis vectors. First observe that one of two sets H_1, H_2 doesn't contain the other since no basis vectors are $\mathbf{0}$ and we have Eqn. (19). Assume without loss of generality that $|H_1| \leq |H_2|$. Let i be $\arg\min_{i' \in H_1 - H_2} k_{i'}$, breaking ties arbitrarily. Then for $j \in H_1 - H_2$ let $k'_j = k_j - k_i$, for $j \in H_2 - H_1$ let $k'_j = k_j + k_i$, and for all other j let $k'_j = k_j$.

Note that there was no change in the sum, i.e.,

$$\begin{aligned} \Psi(w) &= \boldsymbol{v}_0 + k_1 \boldsymbol{v}_1 + \ldots + k_m \boldsymbol{v}_m \\ &= \boldsymbol{v}_0 + k'_1 \boldsymbol{v}_1 + \ldots + k'_m \boldsymbol{v}_m, \end{aligned}$$

which establishes (16).

This new representation $\langle k'_1, k'_2, \ldots, k'_m \rangle$ has a strictly smaller support since $k'_i = 0$; so we have Eqn. (18). Observe that

$$\sum_{j=1}^{m} k'_j - \sum_{j=1}^{m} k_j = -k_i |H_1 \setminus H_2| + k_i |H_2 \setminus H_1| \geq 0$$

since $|H_2| \geq |H_1|$. This gives Eqn. (17).

Thus, we have found another linear representation $\boldsymbol{v}_0 + k'_1 \boldsymbol{v}_1 + \ldots + k'_m \boldsymbol{v}_m$ of $\Psi(w)$ that has a smaller support than the given linear representation $\boldsymbol{v}_0 + k_1 \boldsymbol{v}_1 + \ldots + k_m \boldsymbol{v}_m$ without decreasing the 1-norm value of the linear representation with respect to $\mathcal{L}$. We can repeat this process until we obtain a linear representation of support size at most h.

Finally, recall that we assumed $\lambda \geq 2$. To remove this assumption, as mentioned at the beginning, we can add an unused terminal to Σ, which increments the value of λ by one in the bound. □

6 Bounding the Number of Iterations

This section is devoted to proving Theorem 1.1, restated here.

Theorem 6.1 (Theorem 1.1 Restated). *Let $\mathbb{S}$ be a p-stable commutative semiring. Let P be a* Datalog$^\circ$ *program where the maximum number of multiplicands in any product is at most λ. Let D be the input EDB database. Let σ be the number of the semiring elements referenced in P or D. Let n denote the total number of ground atoms in an IDB that at some point in the iterative evaluation of P*

over semiring $\mathbf{S}$ on input D have a nonzero associated semiring value. Then the iterative evaluation of P over semiring $\mathbf{S}$ on input D converges within

$$\lceil pn(n+3) \cdot (\sigma(n(n+3)/2)\lg(\lambda+1) + 4\sigma \lg \sigma + 1)\rceil$$

steps.

For a vector $\boldsymbol{v} \in \mathbb{N}^\sigma$, we let $Z(\boldsymbol{v})$ denote the product corresponding to $\boldsymbol{v}$, i.e. $\prod_{s=1}^{\sigma} a_s^{v_s}$, where a_s is the element corresponding to the sth entry of the vector. We naturally extend the notation to a vector set $\boldsymbol{V}$, by letting $Z(\boldsymbol{V}) \coloneqq \bigoplus_{\boldsymbol{v}\in\boldsymbol{V}} Z(\boldsymbol{v})$. To prove the theorem, we need the following lemma, which shows that the summation of all products corresponding to vectors in $\mathcal{L}$ with coefficients up to p doesn't change when added a product corresponding to any other vector in $\mathcal{L}$. This lemma can be extracted from the proof of [11] (See Section 5.2, in particular the proof of Theorem 5.10 in the journal / ArXiV version of [11]). Since this paper did not state the lemma exactly as we will use it (the lemma only states that if the underlying semiring is stable, a multivariate polynoial function is stable) and our lemma is not easy to extract from the the previous work at first sight, we include the following lemma and its proof for completeness.

Lemma 6.2 ([11]). *Let $\mathcal{L}$ be a linear set with offset vector $\boldsymbol{v}_0$ and basis vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_m$. Let $\mathcal{L}_{\le p} \coloneqq \{\boldsymbol{v}_0 + \kappa_1\boldsymbol{v}_1 + \kappa_2\boldsymbol{v}_2 + \ldots + \kappa_m\boldsymbol{v}_m \mid \kappa_i \in [0,p] \forall\, i\}$. Consider an arbitrary $\boldsymbol{w} = \boldsymbol{v}_0 + k_1\boldsymbol{v}_1 + k_2\boldsymbol{v}_2 + \ldots + k_m\boldsymbol{v}_m$ where $(k_1, k_2, \ldots, k_m) \in \mathbb{N}^m$ and $k_i > p$ for some i. Then, we have*

$$Z(\mathcal{L}_{\le p}) = Z(\mathcal{L}_{\le p}) \oplus Z(\boldsymbol{w})$$

Proof. Assume wlog that $k_1, \ldots, k_{m'} > p$ and $k_{m'+1}, \ldots, k_m \le p$. Let $G_j \coloneqq \{\boldsymbol{\kappa} = (\kappa_1, \ldots, \kappa_m) \mid \kappa_i \in [0, k_i]\ \forall i \in [0,j] \cup [m'+1, m]$ and $\kappa_i \in [0,p]\ \forall i \in [j+1, m']\}$ for all $j \in [0, m']$. Note that to prove the lemma it suffices to show

$$Z(G_0) = Z(G_0) \oplus Z(\boldsymbol{w})$$

because $\{\boldsymbol{v}_0 + \kappa_1\boldsymbol{v}_1 + \ldots + \kappa_m\boldsymbol{v}_m \mid \boldsymbol{\kappa} \in G_0\} \subseteq \mathcal{L}_{\le p}$. We are going to establish

$$Z(G_0) = Z(G_1) = \ldots = Z(G_{m'}) \tag{27}$$

and

$$Z(G_j) \oplus Z(G_{j+1} \setminus G_j) = Z(G_j)\ \forall j \in [0, m'-1] \tag{28}$$

Indeed if we have them,

$$Z(G_0) \oplus Z(\boldsymbol{w}) = Z(G_{m'-1}) \oplus Z(\boldsymbol{w}) = Z(G_{m'-1}) = Z(G_0),$$

as desired, since $(k_1, k_2, \ldots, k_m) \in G_{m'} \setminus G_{m'-1}$ and $\boldsymbol{w} = \boldsymbol{v}_0 + k_1\boldsymbol{v}_1 + \ldots + k_m\boldsymbol{v}_m$.

It now remains to show Eqn. (27) and (28). Consider a fixed $j \in [0, m'-1]$. Consider an arbitrary $\boldsymbol{\kappa} \in G_j$. Let $\boldsymbol{\kappa}'(q)$ be $\boldsymbol{\kappa}$ with $\boldsymbol{\kappa}_{j+1}$ ($j+1$-th coordinate of $\boldsymbol{\kappa}$) replaced with q. Let $\boldsymbol{v}(\boldsymbol{\kappa}'(q)) \coloneqq \boldsymbol{v}_0 + \kappa'_1(q)\boldsymbol{v}_1 + \ldots + \kappa'_m(q)\boldsymbol{v}_m$ be the vector represented by the linear representation $\boldsymbol{\kappa}'(q)$. Let $z_i \coloneqq Z(\boldsymbol{v}_i)$. Then, we have

$$\begin{aligned}
\bigoplus_{q=0}^{p} Z(\boldsymbol{v}(\boldsymbol{\kappa}'(q))) &= \bigoplus_{q=0}^{p} \left(\prod_{i=1:i\neq j+1}^{m} z_i^{\kappa_i} \right) z_{j+1}^{q} = \prod_{i=1:i\neq j+1}^{m} z_i^{\kappa_i} \bigoplus_{q=0}^{p} z_{j+1}^{q} \\
&= \prod_{i=1:i\neq j+1}^{m} z_i^{\kappa_i} z_{j+1}^{(p)} \\
&= \prod_{i=1:i\neq j+1}^{m} z_i^{\kappa_i} z_{j+1}^{(k_{j+1})} \quad [p\text{-stability and } k_{j+1} > p]
\end{aligned}$$

$$= \bigoplus_{q=0}^{p} \left(\prod_{i=1:i\neq j+1}^{m} z_i^{\kappa_i} \right) z_{j+1}^{q} + \bigoplus_{q=p+1}^{k_{j+1}} \left(\prod_{i=1:i\neq j+1}^{m} z_i^{\kappa_i} \right) z_{j+1}^{q}$$

$$= \bigoplus_{q=0}^{p} Z(\boldsymbol{v}(\boldsymbol{\kappa}'(q))) + \bigoplus_{q=p+1}^{k_{j+1}} Z(\boldsymbol{v}(\boldsymbol{\kappa}'(q)))$$

Since $\bigcup_{q=0}^{p} \boldsymbol{v}(\boldsymbol{\kappa}'(q)) \subseteq G_j$, and any vector in $G_{j+1} \setminus G_j$ is of the form of $\boldsymbol{\kappa}'(q)$ for some $q \in [p+1, k_{j+1}]$ for some $\boldsymbol{\kappa} \in G_j$, we have $Z(G_j) = Z(G_{j+1})$. In other words, we showed that any product corresponding to $G_{j+1} \setminus G_j$ is subsumed by some $p+1$ products in $Z(G_j)$ using the p-stability. For the same reason, we have Eqn. (28). □

We now have all tools to prove Theorem 1.1. Consider an arbitrary IDB variable X_r and let L be the CFL associated with this variable. Let $\mathcal{M}$ be a semi-linear set that satisfies the properties stated in Theorem 3.1. Let $\mathcal{T}_q^r$ denote the collection of the parse trees of depth at most q starting with X_r. Our goal is to show:

$$f_r^{(q)}(\mathbf{0}) = f_r^{(q+1)}(\mathbf{0}) \tag{29}$$

where

$$f_r^{(q)}(\mathbf{0}) = \bigoplus_{T \in \mathcal{T}_q^r} Z(T), \text{ and}$$

$$q := \lceil pn(n+3) \cdot (\sigma(n(n+3)/2) \lg(\lambda+1) + 4\sigma \lg \sigma + 1) \rceil \tag{30}$$

Consider an arbitrary $T \in \mathcal{T}_{q+1}^r \setminus \mathcal{T}_q^r$. By Theorem 3.1 (2)(b), $Y(T)$ has a linear representation

$$\Psi(Y(T)) = \boldsymbol{v}_0 + k_1\boldsymbol{v}_1 + k_2\boldsymbol{v}_2 + \ldots + k_m\boldsymbol{v}_m$$

within some $\mathcal{L}$ in $\mathcal{M}$ such that $1 + k_1 + k_2 + \ldots + k_m > q/(n(n+3)/2)$. Thus, we have $k := k_1 + k_2 + \ldots + k_m > ph$ where

$$h := 2(\sigma(n(n+3)/2) \lg(\lambda+1) + 4\sigma \lg \sigma)$$

By Lemma 3.2, we can find a linear representation $\Psi(Y(T)) = \boldsymbol{v}_0 + k_1'\boldsymbol{v}_1' + k_2'\boldsymbol{v}_2 + \ldots + k_h'\boldsymbol{v}_h'$, where $k_1' + k_2' + \ldots + k_h' > ph$. By the pigeonhole's principle, we have that $k_j' > p$ for some j.

Let $\mathcal{L}'$ be the subset of $\mathcal{L}$ that only consists of basis vectors $\boldsymbol{v}_1', \boldsymbol{v}_2', \ldots, \boldsymbol{v}_h'$ together with offset vector $\boldsymbol{v}_0$. Clearly, $\mathcal{L}'$ is a linear set. Then, by Lemma 6.2, we have

$$\bigoplus_{\boldsymbol{u} \in \mathcal{L}'_{\leq p}} Z(\boldsymbol{u}) = \bigoplus_{\boldsymbol{u} \in \mathcal{L}'_{\leq p}} Z(\boldsymbol{u}) \oplus Z(T),$$

where we used $\bigoplus_{\boldsymbol{u} \in \mathcal{L}'_{\leq p}} Z(\boldsymbol{u}) = Z(\mathcal{L}'_{\leq p})$, which is the case by definition. To complete the proof of Theorem 1.1, it is sufficient to show

$$\mathcal{L}'_{\leq p} \subseteq \{\Psi(Y(T)) \mid T \in \mathcal{T}_q^r\} \tag{31}$$

To see this consider any $\boldsymbol{v} = \boldsymbol{v}_0 + \kappa_1\boldsymbol{v}_1 + \ldots + \kappa_h\boldsymbol{v}_h \in \mathcal{L}'_{\leq p}$. By definition of $\mathcal{L}'_{\leq p}$, $\kappa_i \leq p$ for all $i \in [h]$. Then, thanks to Theorem 3.1 (2)(a), we know that there is a word $w \in L$ with $\Psi(w) = \boldsymbol{v}$ such that w is generated by a parse tree T' of depth at most $(ph+1)n(n+3)/2 \leq q$. Thus, it must be the case that $\boldsymbol{v} \in \{\Psi(Y(T)) \mid T \in \mathcal{T}_q^r\}$. This establishes Eqn. (31) as desired, and therefore we have proven Theorem 1.1.

7 Conclusion

This paper considers the convergence of recursive Datalog° programs using natural iterative evaluation over the semirings over stable commutative semirings, where convergence is not program dependent. Previously the best-known bound on convergence time was exponential in the output size. Our main contribution is to show that in fact the time to convergence can be bounded by a polynomial in the natural parameters, such as the output size. One consequence of this result is a better understanding of how much worse the time to convergence can be for general Datalog° programs than linear Datalog° programs. One reasonable interpretation of our results is that the worst-case time to convergence for general Datalog° programs is not too much worse than the worst-case time to convergence for linear Datalog° programs, which was a bit surprising to us given that generally one doesn't expect algorithmic convergence bounds for non-linear optimization to be competitive with the bounds for linear optimization.

There are several natural directions for followup research. While essentially tight bounds are known for convergence time for linear Datalog° programs, we do not establish the tightness of our bound. So one natural research direction is to determine tight bounds on the convergence rate for general Datalog° programs. Another natural research direction is to show certain bounds on convergence time over non-stable semirings. Note that such bounds would have to be program-dependent. A further research direction would be to develop other algorithms for evaluating Datalog° programs and analyze their convergence bounds.

Acknowledgments

Sungjin Im was supported in part by NSF grants CCF-2423106, CCF-2121745, and CCF-1844939, and Office of Naval Research Award N00014-22-1-2701. Benjamin Moseley was supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair, NSF grants CCF-2121744 and CCF-1845146, and Office of Naval Research Award N00014-22-1-2702. Kirk Pruhs was supported in part by the NSF grants CCF-2209654 and CCF-1907673, and an IBM Faculty Award. Part of this work was conducted while the authors participated in the Fall 2023 Simons Program on Logic and Algorithms in Databases and AI.

References

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

[2] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. 2016. FAQ: Questions Asked Frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Tova Milo and Wang-Chiew Tan (Eds.). ACM, 13–28. https://doi.org/10.1145/2902251.2902280

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to algorithms* (third ed.). MIT Press, Cambridge, MA. xx+1292 pages.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, 3rd Edition.* MIT Press. http://mitpress.mit.edu/books/introduction-algorithms

[5] Javier Esparza, Stefan Kiefer, and Michael Luttenberger. 2010. Newtonian program analysis. *J. ACM* 57, 6 (2010), 33:1–33:47. https://doi.org/10.1145/1857914.1857917

[6] Robert W. Floyd. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5, 6 (1962), 345. https://doi.org/10.1145/367766.368168

[7] Michel Gondran and Michel Minoux. 2008. *Graphs, dioids and semirings.* Operations Research/Computer Science Interfaces Series, Vol. 41. Springer, New York. xx+383 pages. New models and algorithms.

[8] Todd J. Green, Gregory Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, Leonid Libkin (Ed.). ACM, 31–40. https://doi.org/10.1145/1265530.1265535

[9] Mark W. Hopkins and Dexter Kozen. 1999. Parikh's Theorem in Commutative Kleene Algebra. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999.* IEEE Computer Society, 394–401. https://doi.org/10.1109/LICS.1999.782634

[10] Sungjin Im, Benjamin Moseley, Hung Q. Ngo, and Kirk Pruhs. 2024. On the Convergence Rate of Linear Datalog ˆ∘ over Stable Semirings. In *27th International Conference on Database Theory, ICDT 2024, March 25-28, 2024, Paestum, Italy (LIPIcs, Vol. 290)*, Graham Cormode and Michael Shekelyan (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 11:1–11:20. https://doi.org/10.4230/LIPICS.ICDT.2024.11

[11] Mahmoud Abo Khamis, Hung Q. Ngo, Reinhard Pichler, Dan Suciu, and Yisu Remy Wang. 2022. Convergence of Datalog over (Pre-) Semirings. In *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Leonid Libkin and Pablo Barceló (Eds.). ACM, 105–117. https://doi.org/10.1145/3517804.3524140

[12] Dexter C. Kozen. 1997. *Automata and Computability*. Springer-Verlag, Berlin, Heidelberg.

[13] Werner Kuich. 1997. Semirings and formal power series: their relevance to formal languages and automata. In *Handbook of formal languages, Vol. 1*. Springer, Berlin, 609–677.

[14] Daniel J. Lehmann. 1977. Algebraic Structures for Transitive Closure. *Theor. Comput. Sci.* 4, 1 (1977), 59–76. https://doi.org/10.1016/0304-3975(77)90056-1

[15] Harry R. Lewis and Christos H. Papadimitriou. 1998. *Elements of the theory of computation, 2nd Edition.* Prentice Hall.

[16] Rohit J. Parikh. 1966. On context-free languages. *J. Assoc. Comput. Mach.* 13 (1966), 570–581. https://doi.org/10.1145/321356.321364

[17] D. L. Pilling. 1973. Commutative regular equations and Parikh's theorem. *J. London Math. Soc. (2)* 6 (1973), 663–666. https://doi.org/10.1112/jlms/s2-6.4.663

[18] Günter Rote. 1985. A systolic array algorithm for the algebraic path problem (shortest paths; Matrix inversion). *Computing* 34, 3 (1985), 191–219. https://doi.org/10.1007/BF02253318

[19] Günter Rote. 1990. Path problems in graphs. In *Computational graph theory*. Comput. Suppl., Vol. 7. Springer, Vienna, 155–189. https://doi.org/10.1007/978-3-7091-9076-0_9

[20] Richard P. Stanley. 2012. *Enumerative combinatorics. Volume 1* (second ed.). Cambridge Studies in Advanced Mathematics, Vol. 49. Cambridge University Press, Cambridge. xiv+626 pages.

[21] Robert Endre Tarjan. 1981. A Unified Approach to Path Problems. *J. ACM* 28, 3 (1981), 577–593. https://doi.org/10.1145/322261.322272

[22] Stephen Warshall. 1962. A Theorem on Boolean Matrices. *J. ACM* 9, 1 (1962), 11–12. https://doi.org/10.1145/321105.321107

Received May 2024; revised August 2024; accepted September 2024