Convergence of Datalog over (Pre-) Semirings

Mahmoud Abo Khamis
relationalAI

Hung Q. Ngo
relationalAI

Reinhard Pichler
TU Wien

Dan Suciu
University of Washington

Yisu Remy Wang
University of Washington

ABSTRACT

Recursive queries have been traditionally studied in the framework of datalog, a language that restricts recursion to monotone queries over sets, which is guaranteed to converge in polynomial time in the size of the input. But modern big data systems require recursive computations beyond the Boolean space. In this paper we study the convergence of datalog when it is interpreted over an arbitrary semiring. We consider an ordered semiring, define the semantics of a datalog program as a least fixpoint in this semiring, and study the number of steps required to reach that fixpoint, if ever. We identify algebraic properties of the semiring that correspond to certain convergence properties of datalog programs. Finally, we describe a class of ordered semirings on which one can generalize the semi-naïve evaluation algorithm to compute their minimal fixpoints.

1. INTRODUCTION

Traditional database systems have focused on non-recursive (loop-free) queries. However, modern data processing and tensor computations require iteration and fixpoint computation. Some systems, like Spark [33], have iteration embedded natively, while others, like Tensorflow [1] are routinely run by a driver, e.g. in order to iterate the computation until convergence.

Theoretically, the database community has studied the evaluation and optimization problem for iterative programs in the context of datalog [2], a language that restricts recursion to monotone queries over sets. However, today's applications need to iterate expressions that are not monotone queries over sets. For example standard tensor operations (e.g. Einsum, convolution) are not monotone w.r.t. set inclusion. Even some purely relational problems such as computing *betweenness centrality* [11] or *all-pairs shortest paths* (APSP) [5] require algebraic computations, and thus are not monotone in the datalog-sense.

Recursive computation beyond the Boolean case leads to three major challenges. First, non-monotonicity of general aggregations leads to difficult semantic conundra [25, 24]. Second, finite convergence behavior of the computation is no longer easy to be swept under the rug (see examples below). Third, out of the confine of Boolean monotonicity, optimization techniques developed for datalog such as semi-naïve evaluation [2] no longer work.

Suciu and Wang were partially supported by NSF IIS 1907997 and NSF IIS 1954222. Pichler was supported by the Austrian Science Fund (FWF):P30930 and by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT2201]

To address the demand for more expressive computations, this paper introduces the language datalog$^\circ$, pronounced *datalogo*, which allows for expressing recursive computations over general semirings; the superscript "$\circ$" is a (semi)-ring. Datalog$^\circ$ is powerful in that it can express problems such as transitive closure, all-pairs shortest-paths (APSP), minimum spanning tree (MST), or computing a local minimum of a (class of) optimization problem(s). To address the aforementioned challenges, we equip datalog$^\circ$ with a natural (least fixpoint) semantics, study convergence behavior of datalog$^\circ$ programs, and generalize semi-naïve evaluation to work over semirings (modulo specific assumptions). In what follows, we *informally* describe datalog$^\circ$ and how we took steps towards addressing the challenges.

Expressiveness. A datalog program is a collection of (unions of) conjunctive queries, operating on relations. Analogously, a datalog$^\circ$ program is a collection of (sum-) sum-product queries over a (pre-) semiring, operating on S-relations. An S-relation is a function from the set of tuples to a semiring S, which is a set equipped with two operations, $\oplus$ and $\otimes$, and relational algebra extends naturally to S-relations.

EXAMPLE 1.1. *Let A be a real-valued matrix. A is an $\mathbb{R}$-relation, where each tuple $A(i,j)$ has the value a_{ij}; and, $\mathbb{R}$ denotes the sum-product semiring $(\mathbb{R}, +, \cdot, 0, 1)$. Both the objective and gradient of the* ridge linear regression *problem $\min_x J(x)$, with $J(x) = \frac{1}{2}\|Ax-b\|^2 + (\lambda/2)\|x\|^2$, are expressible in datalog$^\circ$, because they are sum-sum-product queries. The gradient $\nabla J(x) = A^\top Ax - A^\top b + \lambda x$, for example, is the following sum-sum-product query:*

$$\nabla(i) = \sum_j \sum_k a(k,i) \cdot a(k,j) \cdot x(j) + \sum_j (-1) \cdot a(j,i) \cdot b(j) + \lambda() \cdot x(i)$$

The gradient has the same dimensionality as x; the group by variable is i. Gradient descent is an algorithm to solve for the solution of $\nabla J(x) = 0$, or equivalently to solve for

The notion of *K-relations* was introduced by Green et al. [13]; we prefer to call them *S-relations* in this paper where S stands for "semiring".

a fixpoint solution to the datalog$^\circ$ *program* $x = f(x)$ *where* $f(x) = \nabla J(x) + x$.

Example 1.2. *The APSP problem is to compute the shortest path length* $P(x, y)$ *between any pair* x, y *of vertices in the graph, given the length* $E(x, y)$ *of edges in the graph. The value-space of* $E(x, y)$ *can be the reals* $\mathbb{R}$ *or the non-negative reals* $\mathbb{R}_+$. *The APSP problem in* datalog$^\circ$ *can be expressed very compactly as*

$$P(x, y) \text{ :- } E(x, y) \oplus \bigoplus_z P(x, z) \otimes E(z, y), \quad (1)$$

where $(\oplus, \otimes) = (\min, +)$ *are the "addition" and "multiplication" operators in one of the* $(\min, +)$ *tropical semirings* $\mathsf{Trop} := (\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$, *or* $\mathsf{Trop}^+ := (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$.

By changing the semiring, datalog$^\circ$ *is able to express similar problems in exactly the same way. For example,* (1) *becomes* transitive closure *over the Boolean semiring,* top p-shortest-paths *over the* Trop_p^+ *(Sec. 5) and so forth.*

Semantics. It should be clear that datalog$^\circ$ is very powerful. Unfortunately, "*with great power comes great responsibility*". In datalog, the least fixpoint semantics was defined w.r.t. set inclusion [2]. Generalizing to semirings, Green et al.[13] observed that, in order to define the semantics of datalog over S-relations, one needs a *partial order*, $\sqsubseteq$, because the least fixpoint is defined w.r.t. some partial order. They proposed to use semirings which are *naturally ordered*, where $x \sqsubseteq y$ is defined as $\exists z : x + z = y$. However, some important semirings are not naturally ordered. For example, $(\mathbb{R}, +, \cdot, 0, 1)$ is not naturally ordered, because the relation $\exists z : x + z = y$ is *not* anti-symmetric: for any $x \neq y$ there is a z where $x + z = y$ and $y + (-z) = x$. This means that recursive programs over arrays, matrices, or tensors cannot be interpreted using the framework in [13].

In datalog$^\circ$ we *decouple* the semiring structure from the partial order. We define a *partially ordered, pre-semiring*, denoted by POPS, to be any pre-semiring with a partial order, where both $\oplus$ and $\otimes$ are monotone operations. The value-space of every S-relation is some partially ordered pre-semiring.

In some cases, e.g. the Booleans, the partial order of POPS is the natural one, but in other cases it is not. For example, we can define a non-standard order relation on $\mathbb{R}$ by adding a new element, $\mathbb{R}_\perp \overset{\text{def}}{=} \mathbb{R} \cup \{\perp\}$, and defining $\perp \sqsubseteq x$ for all x. The set $\mathbb{R}_\perp$ is called the *lifted reals*, and we can use it to define a semantics for recursive programs over vectors, matrices, or tensors. Adding $\perp$ to create a partial order is the standard approach for defining the semantics in general programming languages [29]. In logic programming, Fitting [10] proposed adding $\perp$, leading to the 3-*valued semantics* of logic programs *with* negation.

With the partial order in place, we define the semantics of a datalog$^\circ$ program as the least fixpoint of the immediate consequence operator. Datalog$^\circ$ subsumes traditional datalog semantics, *and* captures the semantics of complex, recursive computations over vectors, matrices, tensors, etc.

A pre-semiring is a semiring without the absorption axiom $x \otimes 0 = 0 \otimes x = 0$ [12]; see Sec. 2.

Finite Convergence. Example 1.1 hinted at the difficulty with computing the *exact* least fixpoint solution in a general POPS, even when we know that the fixpoint exists. In practice, numerical optimization problems are often only solved approximately. This, in principle, remains true with datalog$^\circ$. However, here we concentrate on ways to compute the *exact* least fixpoint solution in a *finite* number of steps. In particular, we focus on analyzing the number of iterations needed for the naïve evaluation algorithm to converge.

Note that the infinite cardinality of the semiring value-space is *not* the reason why a datalog$^\circ$ program does not converge when iterated naïvely. For example, intuitively we know that the naïve algorithm for the APSP program (1) may not converge under Trop (due to negative cycles), but it will always converge under Trop^+. This paper makes this intuition precise: we extract algebraic properties of the POPS that are necessary and sufficient conditions under which the naïve algorithm for datalog$^\circ$ converges in a finite number of steps. Furthermore, under additional assumptions, we show that the naïve algorithm converges in poly-time. These results subsume the corresponding results in traditional datalog.

For example, our results imply that while the naïve algorithm for (1) may not converge under Trop, it converges in a linear number of steps if the semiring is Trop^+ or more generally Trop_p^+. The property satisfied by Trop^+ but not Trop is $\mathbf{1} \oplus a = \mathbf{1}$ for every element a in the value-space, where $\mathbf{1}$ denotes the multiplicative identity. Less formally, this says $\min(a, 0) = 0$ for all $a \in \mathbb{R}_+ \cup \{\infty\}$.

Beyond the naïve algorithm, we show that the approach of generalizing Gaussian elimination to *closed* semirings [28, 21] works for datalog$^\circ$ under POPS too; this leads to essentially a cubic time algorithm to find a least fixpoint of a linear datalog$^\circ$ program.

Optimization. Semi-naïve evaluation is one of the major optimization techniques for datalog, which we would like to generalize to datalog$^\circ$. To explain the main ideas, let us consider the Boolean semiring version of (1):

$$P(x, y) \text{ :- } E(x, y) \vee \bigvee_z P(x, z) \wedge E(z, y). \quad (2)$$

After initializing $\delta^{(0)}(x, y) = P^{(1)}(x, y) = E(x, y)$, at iteration $t \geq 1$ semi-naïve evaluation does the following:

$$\delta^{(t)}(x, y) = \left(\bigvee_z \delta^{(t-1)}(x, z) \wedge E(z, y)\right) \setminus P^{(t)}(x, y) \quad (3)$$
$$P^{(t+1)}(x, y) = P^{(t)}(x, y) \cup \delta^{(t)}(x, y).$$

Without this optimizations, we will have rederived a lot of facts in each iteration. For certain semirings, we are able to generalize the above ideas to datalog$^\circ$, by defining an appropriate "minus" operator, denoted $\ominus$, which plays the role of the $\setminus$ operator in (3). Then, we show that the semi-naïve evaluation (3) holds for general datalog$^\circ$ programs.

Example 1.3 (APSP). *The APSP problem is* (1) *under* Trop, *where the analog of* (3) *is:*

$$\delta^{(t)}(x, y) = (\min_z \delta^{(t-1)}(x, z) + E(z, y)) \ominus P^{(t)}(x, y),$$

Repeatedly apply the equation until a fixpoint is reached *where the difference operator $\ominus$ is:*

$$v \ominus u = \begin{cases} v & \text{if } v < u \\ \infty & \text{if } v \geq u \end{cases} \tag{4}$$

Intuitively, the $\ominus$ operator does what we expect: from the new shortest paths from x to z discovered in the previous iteration, we see if adding a (z, y) edge gives us a shorter path from x to y; if so, we update the shortest path from x to y.

Paper organization. Section 2 presents basic concepts on semirings and defines (sum-)sum-product queries under semirings, and Section 3 reviews the concept of a least fixpoint. Section 4 formulates the concept of partially ordered pre-semirings (POPS), which form the value-spaces of datalog$^\circ$ programs. Then, it formally defines the syntax for datalog$^\circ$ programs, and its fixpoint semantics. We present our fundamental results on the convergence behaviors of datalog$^\circ$ programs in Section 5. Section 6 describes a generalization of semi-naïve evaluation to datalog$^\circ$. Section 7 outlines how datalog$^\circ$ is used in practice and the challenges that arise. Finally, we conclude in Sec. 8.

This paper is a shortened version of the PODS'2022 paper [16]. The statement of main result Theorem 5.2 is improved over the conference version, and its proof is given in the full version [15]. We also include here the new Section 7 on practical considerations.

2. BACKGROUND ON SEMIRINGS

A *pre-semiring* [12] is a tuple $\boldsymbol{S} = (S, \oplus, \otimes, 0, 1)$ where $\oplus$ and $\otimes$ are binary operators on S for which $(S, \oplus, 0)$ is a commutative monoid, $(S, \otimes, 1)$ is a monoid, and $\otimes$ distributes over $\oplus$. This paper only considers *commutative* $\oplus$ and $\otimes$ operators. When the *absorption rule* $x \otimes 0 = 0$ holds for all $x \in S$, we call $\boldsymbol{S}$ a *semiring.*

Examples are the Boolean semiring ($\mathbb{B} \stackrel{\text{def}}{=} \{0, 1\}, \vee, \wedge, 0, 1$), and sum-product semirings over the naturals $(\mathbb{N}, +, \cdot, 0, 1)$ or reals $(\mathbb{R}, +, \cdot, 0, 1)$. We will refer to them simply as $\mathbb{B}, \mathbb{N}$ and $\mathbb{R}$. Other useful examples were introduced in Example 1.2: Trop and Trop$^+$. We will illustrate more semirings in this paper, and also refer the reader to [12] and the full version of this paper [15] for additional examples.

Fix a pre-semiring $\boldsymbol{S} = (S, \oplus, \otimes, 0, 1)$ and a domain D. To simplify the discussion we will assume that D is finite, and defer the general case to [15]. For example D could be the set $[n] = \{1, \ldots, n\}$, or some finite set of identifiers. An $\boldsymbol{S}$*-relation* is a function $R : D^k \to S$. The domain D is called the *key-space*, S is the *value-space*, and k is the arity of R. Fix a a semiring $\boldsymbol{S}$, and a vocabulary $\{R_1, R_2, \ldots\}$ of relation names, where each relation R_j has type $D^{k_j} \to S$, with $k_j =$ the arity of R_j.

Definition 2.1. *Let $x_1, \ldots, x_p$ be a set of variables, taking values in the domain D. A* sum-product *query is an expression of the form*

$$T(x_1, \ldots, x_k) \text{ :- } \bigoplus_{x_{k+1}, \ldots, x_p \in D} A_1 \otimes \cdots \otimes A_m \tag{5}$$

where each A_u is either a relational atom, *$R_i(x_{t_1}, \ldots, x_{t_{k_i}})$, or an equality predicate, $[x_t = x_s]$; the variables $x_1, \ldots, x_k$ are called* free variables*, and the others are called* bound variables*. The body of the query (RHS of (5)) is called a* sum-product expression*.*

A sum-sum-product *query has the form:*

$$Q(x_1, \ldots, x_k) \text{ :- } T_1(x_1, \ldots, x_k) \oplus \cdots \oplus T_q(x_1, \ldots, x_k) \tag{6}$$

where $T_1, T_2, \ldots, T_q$ are sum-product expressions with the same free variables $x_1, \ldots, x_k$.

When the value-space $\boldsymbol{S}$ is the Boolean semiring, then a sum-product query is a Conjunctive Query (CQ) under set semantics, and a sum-sum-product query is a Union of Conjunctive Queries (UCQ). When $\boldsymbol{S} = \mathbb{N}$, then they are a CQ or UCQ under bag semantics; and when $\boldsymbol{S} = \mathbb{R}$ then a sum-product expression is a tensor expression, sometimes called an *Einsum* [27].

The problem of computing (sum-)sum-products over semirings efficiently has been extensively studied both in the database and in the AI literature. In databases, the query optimization and evaluation problem is a special case of sum-sum-product computation over the value-space of Booleans (set semantics) or natural numbers (bag semantics). The functional aggregate queries (FAQ) framework [17] extends the formulation to queries over multiple semirings. In AI, this problem was studied by Shenoy and Schafer [30], Dechter [8], Kohlas and Wilson [19] and others. Surveys and more examples can be found in [3, 18].

3. LEAST FIXPOINTS

Fix a *partially ordered set* (poset), $\boldsymbol{L} = (L, \sqsubseteq)$. In this paper we will assume that each poset has a minimum element $\perp$. We denote by $\bigvee A$, or $\bigwedge A$ respectively, the least upper bound, or greatest lower bound of a set $A \subseteq L$, when it exists. A function f between two posets is called *monotone* if $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$ An ω*-chain* in a poset $\boldsymbol{L}$ is a sequence $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$ We say that the chain is finite if there exists n_0 such that $x_{n_0} = x_{n_0+1} = x_{n_0+2} = \cdots$; equivalently, $x_{n_0} = \bigvee x_n$.

Given a monotone function $f : \boldsymbol{L} \to \boldsymbol{L}$, a *fixpoint* is an element x such that $f(x) = x$. We denote by $\mathsf{lfp}(f)$ the *least fixpoint* of f, when it exists. Consider the following ω-sequence:

$$f^{(0)}(\perp) \stackrel{\text{def}}{=} \perp \qquad f^{(n+1)}(\perp) \stackrel{\text{def}}{=} f(f^{(n)}(\perp)) \tag{7}$$

When the ω-chain (7) is finite, then its last element is the least fixpoint of f. Otherwise, the least fixpoint may, or may not exists. Green et al. [13] consider semirings that are ω-continuous, and, in that case, the least fixpoint is guaranteed to exists and to be equal to $\bigvee_{n \geq 0} f^{(n)}(0)$. We do not consider ω-continuous semirings in this paper.

A poset $\boldsymbol{L}$ satisfies the *Ascending Chain Condition*, or ACC [26], if it has *no infinite ω-chains.* It follows that ACC is a sufficient condition for the existence of a least fixpoint for any monotone function f.

4. DATALOG$^\circ$

We introduce here a new recursive language obtained by combining traditional datalog with $\boldsymbol{S}$-relations. We call the language datalog$^\circ$, pronounced *datalogo*, where the superscript "$\circ$" is a (semi)-ring. We restrict our discussion to *basic* datalog$^\circ$, which is analogous to the monotone fragment of datalog without interpreted functions, and discuss extensions

Some references, e.g. [19], define a semiring without absorption.

in our full paper [15]. We denote by $E_1, \ldots, E_m$ the relational symbols representing the EDB predicates, and by $T_1, \ldots, T_n$ the symbols representing the IDB predicates. All EDBs and IDBs are assumed to be over the same finite domain D, and have the same value-space $\boldsymbol{S}$. A $\mathsf{datalog}^\circ$ program Π consists of n rules, one for each IDB predicate:

$$\begin{aligned} T_1(\mathtt{vars}_1) &\text{ :- } F_1(E_1, \ldots, E_m, T_1, \ldots, T_n) \\ &\cdots \\ T_n(\mathtt{vars}_n) &\text{ :- } F_n(E_1, \ldots, E_m, T_1, \ldots, T_n) \end{aligned} \tag{8}$$

where each function $F_1, \ldots, F_n$ is a sum-sum-product expression given by Eq. (6). P is called *linear* if each product contains at most one IDB predicate. The $\mathsf{datalog}^\circ$ program in Ex. 1.2 is linear.

Partially Ordered Pre-semirings. To define the semantics of $\mathsf{datalog}^\circ$, we extend the pre-semiring $\boldsymbol{S}$ with a partial order.

Definition 4.1. *A partially ordered pre-semiring (POPS) is a tuple $\boldsymbol{P} = (P, \oplus, \otimes, 0, 1, \sqsubseteq)$, where $(P, \oplus, \otimes, 0, 1)$ is a pre-semiring, $(P, \sqsubseteq)$ is a poset, and $\oplus, \otimes$ are monotone.*

Monotonicity is defined in the usual way: $x \sqsubseteq x'$ and $y \sqsubseteq y'$ implies $x \oplus y \sqsubseteq x' \oplus y'$ and similarly for $\otimes$. Notice that any POPS satisfies the identities $\bot \oplus \bot = \bot$ (because $\bot \oplus \bot \sqsubseteq \bot \oplus 0 = \bot$) and similarly $\bot \otimes \bot = \bot$. We say that $\otimes$ is *strict* if the identity $x \otimes \bot = \bot$ holds for every x. Throughout this paper we require $\otimes$ to be strict.

In any (pre-)semiring $\boldsymbol{S}$, the relation $x \preceq_S y$ defined as $\exists z : x \oplus z = y$, is a *preorder*: it is reflexive and transitive, but not always anti-symmetric. When $\preceq_S$ is anti-symmetric, it is a partial order, in which case it is called the *natural order* on $\boldsymbol{S}$. Every naturally ordered semiring is a POPS, where $\bot = 0$ and $\otimes$ is strict, but not conversely. Our approach differs from the setting in Green at al. [13] and Dannert et al. [7], who required the semiring to be naturally ordered. In other words, we decouple the order relation from the (pre-)semiring structure. The reason is that many $\mathsf{datalog}^\circ$ programs are over a semiring that is not naturally ordered, and, even when it is, their fixpoint semantics is over an order different from the natural one. For example, $\mathbb{R}$ is not naturally ordered, yet all tensor operations are over $\mathbb{R}$-relations, hence we need a means to interpret recursive programs over $\mathbb{R}$. The common approach for defining recursive functions over $\mathbb{R}$ or $\mathbb{N}$ (or any other set) is to "lift" the set. For example, the lifted integers $\mathbb{N}_\bot \stackrel{\text{def}}{=} \mathbb{N} \cup \{\bot\}$ with the ordering $\bot \sqsubseteq x$ for all x, is used to give a standard textbook semantics to the recursive definition of factorial:

$$P: \quad \mathtt{fact}(x) = \mathtt{if}(x = 0) \ \mathtt{then} \ 1 \ \mathtt{else} \ x \cdot \mathtt{fact}(x - 1)$$

One can view P as a function that takes as input some partial function $\mathtt{fact}$ and returns a new partial function, namely $\mathtt{if}(x = 0)$ $\mathtt{then}$ 1 $\mathtt{else}$ $x \cdot \mathtt{fact}(x - 1)$. If one applies P q times to $\bot$ (the totally undefined function), the result $P^{(q)}(\bot)$ is the function $x!$ for $x < q$, and $\bot$ for $x \geq q$. The standard factorial function is $\mathsf{lfp}(P)$. Important for our discussion is the fact that the partial order $\sqsubseteq$ for which we define the least fixpoint is not the natural order $x \leq y$ on $\mathbb{N}$, but a different one. This justifies our definition of a POPS (Def 4.1), where we decoupled the partial order from the (pre-)semiring operations.

EDB and IDB stand for extensional database and intentional database respectively [2].

Semantics. Fix a POPS $\boldsymbol{P}$ and consider a $\mathsf{datalog}^\circ$ program (8) over $\boldsymbol{P}$, meaning that its EDB and IDB relations are $\boldsymbol{P}$-relations. Let I be an instance of all EDB relations $E_1, \ldots, E_m$ and J be an instance of the IDB relations $T_1, \ldots, T_n$. Each rule $T_j(\mathtt{vars}_1)$:- $F_j(\cdots)$ in (8) computes a new state of the IDB T_j from I and J. The *immediate consequence operator* is the function $F(I, J)$ that evaluates all n rules in (8) on I and J, and returns the n-tuple of new IDB values $J' = F(I, J)$. The *semantics* of the $\mathsf{datalog}^\circ$ program (8) is the least fixpoint of the mapping $J \mapsto F(I, J)$, i.e. $\mathsf{lfp}(\lambda J.F(I, J))$, when it exists, and undefined otherwise.

Algorithm 1 is called the *naïve evaluation algorithm* for $\mathsf{datalog}^\circ$. It computes the ω-sequence $J^{(0)} \sqsubseteq J^{(1)} \sqsubseteq \cdots$ of IDBs. When the sequence is finite, then the algorithm returns the last value $J^{(t)}$; in that case we say that the algorithm *converges*. Otherwise, we say that the algorithm *diverges*. Notice that the first line $J^{(0)} \leftarrow \bot$ becomes $J^{(0)} \leftarrow 0$ when the semiring is naturally ordered. (Of course, $J^{(0)} \leftarrow \bot$ means that $J^{(0)}$ is the $\boldsymbol{P}$-instance that maps every ground atom to $\bot$.)

Algorithm 1: Naïve evaluation for $\mathsf{datalog}^\circ$

```
$J^{(0)} \leftarrow \bot$;                // or $J^{(0)} \leftarrow 0$, see text
for $t \leftarrow 0$ to $\infty$ do
    $J^{(t+1)} \leftarrow F(J^{(t)})$;
    if $J^{(t+1)} = J^{(t)}$ then
        Break
return $J^{(t)}$
```

5. CONVERGENCE BEHAVIOR

In traditional datalog, the naïve evaluation algorithm always converges in a number of iterations that is polynomial in the size of the input database, but in $\mathsf{datalog}^\circ$, the picture is more complex. Recall that D denotes the (finite) domain representing the key-space, and $\boldsymbol{P}$ is a POPS representing the value space. Depending on the POPS $\boldsymbol{P}$, there are five cases. For any $\boldsymbol{P}$ instance I, we denote by $\mathsf{ADom}(I)$ the set of tuples whose value is $\neq \bot$.

(i) For some $\mathsf{datalog}^\circ$ programs, $\bigvee_t J^{(t)}$ is not the least fixpoint.

(ii) Every $\mathsf{datalog}^\circ$ program has the least fixpoint $\bigvee_t J^{(t)}$, but may not necessarily converge.

(iii) Every $\mathsf{datalog}^\circ$ program converges.

(iv) Every $\mathsf{datalog}^\circ$ program converges in a number of steps that depends only on $|\mathsf{ADom}(I)|$.

(v) Every $\mathsf{datalog}^\circ$ program converges in a number of steps that is polynomial in $|\mathsf{ADom}(I)|$.

In this paper, we will only consider data-complexity [31], where the $\mathsf{datalog}^\circ$ program is assumed to be fixed, and the input consists only of the EDB instance I.

All POPS that we have encountered in applications are ω-continuous (see [7] for a definition), and this implies that $\bigvee_t J^{(t)}$ is always the least fixpoint; for that reason we will no longer discuss case (i). We start by illustrating how the choice of POPS can separate case (ii) from the rest.

We originally described four cases in [16], and refined it to five in the full version [15].

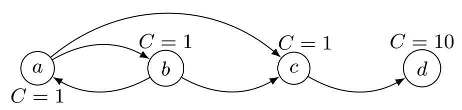

Figure 1: A graph illustrating Example 5.1

Example 5.1. *A classic problem that requires the interleaving of recursion and aggregation is the bill-of-material (see, e.g., [34]). We are given a set of parts, every part X has an intrinsic cost $C(X)$, and $E(X, Y)$ denotes the fact that part X has subpart Y. We are asked to compute, for each part X, its total cost, consisting of its own cost and that of all its sub-parts, sub-sub-parts, etc. The* datalog$^\circ$ *program is:*

$$T(X) \text{ :- } C(X) + \sum_{Y} \{T(Y) \mid E(X, Y)\} \tag{9}$$

When the graph defined by E is a tree then the program computes correctly the bill-of-material. We are interested, however, in what happens when the graph encoded by E has cycles, as illustrate with Fig. 1. The grounded program is:

$$\begin{aligned} T(a) &\text{ :- } C(a) + T(b) + T(c)\\ T(b) &\text{ :- } C(b) + T(a) + T(c)\\ T(c) &\text{ :- } C(c) + T(d)\\ T(d) &\text{ :- } C(d) \end{aligned}$$

We consider two choices for the POPS. First, the naturally ordered semiring $(\mathbb{N}, +, \cdot, 0, 1)$. Thus, both $C(X)$ and $T(X)$ are natural numbers. Then the program diverges, since the naïve algorithm will compute ever increasing values for $T(a)$ and $T(b)$, which lie on a cycle. Second, consider the lifted reals $\mathbb{R}_\perp = (\mathbb{R} \cup \{\perp\}, +, \cdot, 0, 1, \sqsubseteq)$. In this case $T(X) = \perp$ is an indication that the value of $T(X)$ is still unknown. Since both $+$ and $\cdot$ are strict, the program converges in 3 steps, as can be seen below:

	$T(a)$	$T(b)$	$T(c)$	$T(d)$
$T^{(0)}$	$\perp$	$\perp$	$\perp$	$\perp$
$T^{(1)}$	$\perp$	$\perp$	$\perp$	10
$T^{(2)}$	$\perp$	$\perp$	11	10
$T^{(3)}$	$\perp$	$\perp$	11	10

As the example illustrates, for some POPS, like $\mathbb{N}$, a datalog$^\circ$ program may diverge, while for other POPS, like $\boldsymbol{R}_\perp$, every datalog$^\circ$ program converges. A sufficient condition for convergence is the ACC. However, it turns out that the ACC is not necessary, for example every datalog$^\circ$ program on Trop$^+$ converges, even though Trop$^+$ does not satisfy ACC. In our extended paper [15] we have completely characterized the POPS that guarantee convergence, and will summarize the results here.

Fix a semiring $\boldsymbol{S}$. For every $c \in \boldsymbol{S}$ and $p \geq 0$ define: $c^{(p)} \overset{\text{def}}{=} 1 \oplus c \oplus c^2 \oplus \cdots \oplus c^p$, where c^k denotes $c \otimes c \otimes \cdots \otimes c$ (k copies of c). An element $c \in \boldsymbol{S}$ is called *p-stable* if $c^{(p)} = c^{(p+1)}$. We say that c is *stable* if there exists p such that c is p-stable. We call the semiring $\boldsymbol{S}$ *stable* if every element is stable, and call it *uniformly stable* if there exists $p \geq 0$ such that every $c \in \boldsymbol{S}$ is p-stable. If $\boldsymbol{S}$ is stable, then it is a naturally ordered semiring, see [12, 15]. We note that 0-stable semirings have been extensively studied, and were called *simple* by Lehmann [21], *c-semirings* by Kohlas [19], and *absorptive* by Dannert et al. [7]. For example, Trop$^+$ is 0-stable.

Consider now a POPS $\boldsymbol{P}$, where multiplication is strict. One can easily check that the set $P \oplus \perp \overset{\text{def}}{=} \{x \oplus \perp \mid x \in P\}$ together with the operations $\oplus, \otimes$ forms a semiring. Our main result from [15] is:

Theorem 5.2. *Given a POPS $\boldsymbol{P}$, the following hold.*

(i) Every datalog$^\circ$ *program converges iff the semiring $\boldsymbol{P} \oplus \perp$ is stable.*

(ii) Every program converges in a number of steps that depends only on $|ADom(I)|$ iff $\boldsymbol{P} \oplus \perp$ is uniformly stable. More precisely, if $\boldsymbol{P} \oplus \perp$ is p-stable, then every datalog$^\circ$ *program converges in $\sum_{i=1}^{N}(p+2)^i$ steps, where N is the number of ground tuples consisting of IDB predicates and constants from $ADom(I)$. Furthermore, if the program is linear, then it converges in $\sum_{i=1}^{N}(p+1)^i$ steps.*

(iii) If $\boldsymbol{P} \oplus \perp$ is 0-stable, then every datalog$^\circ$ *program converges in N steps; in particular, the program runs in polynomial time in the size of the input database.*

The intuition behind the theorem is the following. Consider a cycle of ground IDB atoms in the grounded program. The value of each such atom starts at $\perp$, and will always remain in the semiring $\boldsymbol{P} \oplus \perp$ and, therefore, if the grounded program is cyclic, then convergence is tied to the stability of $\boldsymbol{P} \oplus \perp$. In our example above, $\mathbb{R}_\perp + \perp = \{\perp\}$ is a trivial semiring with one element, and is 0-stable, hence every datalog$^\circ$-program over $\mathbb{R}_\perp$ converges in polynomial time in the size of the input database, by case (iii) of Theorem 5.2.

We illustrate the theorem using a few examples. Notice that when $\boldsymbol{S}$ is a naturally ordered semiring, then $\perp = 0$ and $\boldsymbol{S} \oplus \perp = \boldsymbol{S}$. We start with the tropical semiring Trop$^+$ defined in Sec. 1, which is naturally ordered. We check that Trop$^+$ is 0-stable: indeed, for any element c we have $c^{(0)} = \mathbf{1}_{\mathsf{Trop}^+} = 0$ and $c^{(1)} = \mathbf{1}_{\mathsf{Trop}^+} \oplus_{\mathsf{Trop}^+} c = \min(0, c) = 0$. It follows that every datalog$^\circ$ program over Trop$^+$ also converges.

Next, we describe a semiring that is p-stable but not $p-1$-stable, where p is any natural number. The semiring is denoted Trop$_p^+$, and generalizes Trop$^+$ as follows. Its elements are bags of $p+1$ values in $\mathbb{R}_+ \cup \{\infty\}$. The operator $a \oplus b$ returns the smallest $p+1$ elements of the union of the two bags a, b, and $a \otimes b$ returns the smallest $p+1$ elements of the bag $\{\!\{x + y \mid x \in a, y \in b\}\!\}$. For example, assuming $p = 2$, if $a = \{\!\{4, 4, 8\}\!\}$, $b = \{\!\{5, 6, 6\}\!\}$, then $a \oplus b = \{\!\{4, 4, 5\}\!\}$ and $a \otimes b = \{\!\{9, 9, 10\}\!\}$; we invite the reader to determine the identity elements 0 and 1 in this semiring. One can check that Trop$_p^+$ is p-stable, and is not $p-1$-stable. Observe that if we use this semiring instead of Trop$^+$ in the example (1) in the introduction, then we compute the bag of the $p+1$ shortest paths between any two nodes. This semiring also belongs to case (ii) of Theorem 5.2 and, in particular, every datalog$^\circ$ program over Trop$_p^+$ converges.

Finally, we illustrate briefly a semiring that is non-uniformly stable. Fix some real number $\eta \geq 0$ and call an *η-set* any finite, non-empty set $a \subseteq \mathbb{R}_+ \cup \{\infty\}$ where $\max(a) - \min(a) \leq \eta$. If $a \subseteq \mathbb{R}_+ \cup \{\infty\}$ is a finite, non-empty set, then its η-reduction is the set $\{x \mid x \in a, x - \min(a) \leq \eta\}$; in other

We review the notion of grounded atom and grounded program in [15].

Algorithm 2: Semi-naïve evaluation for datalog$^\circ$

```
$J^{(0)} \leftarrow \mathbf{0}$;
for $t \leftarrow 0$ to $\infty$ do
    $\delta^{(t)} \leftarrow F(I, J^{(t)}) \ominus J^{(t)}$;          // see text
    $J^{(t+1)} \leftarrow J^{(t)} \oplus \delta^{(t)}$;
    if $\delta^{(t)} = \mathbf{0}$ then
        Break
return $J^{(t)}$
```

words, we retain only the elements at most η away from the smallest. The semiring $\mathsf{Trop}^+_{\leq\eta}$ consists of all η-sets, and the following operations: $a \oplus b$ is the η-reduction of $a \cup b$, and $a \otimes b$ is the η-reduction of $\{x + y \mid x \in a, y \in b\}$. One can check that an η-set $a \neq \{0\}$ is $\lceil \frac{\eta}{\min(a-\{0\})} \rceil$-stable, while $\{0\}$ is 0-stable. It follows that $\mathsf{Trop}^+_{\leq\eta}$ is stable, but there is no global p for which all elements are p-stable. Observe that if we use this semiring instead of Trop^+ in the example (1) in the introduction, then we compute the set of all paths between two nodes whose length does not exceed the minimum path by more than η. This semiring belongs to case (i) of Theorem 5.2 and, in particular, every datalog$^\circ$ program over $\mathsf{Trop}^+_{\leq\eta}$ converges.

6. SEMI-NAÏVE OPTIMIZATION

The Naïve Algorithm 1 consists of repeatedly applying the immediate consequence operator to the current state of the IDB relations. This strategy is inefficient because all facts discovered at iteration t will be re-discovered at iterations $t+1, t+2, \ldots$ The *semi-naïve* optimization consists of a modified program that computes $J^{(t+1)}$ by first computing only the "novel" facts $\delta^{(t)} = F(J^{(t)}) - J^{(t)}$, which are then added to $J^{(t)}$ to form $J^{(t+1)}$. Furthermore, the difference $F(J^{(t)}) - J^{(t)}$ can be computed efficiently, *without* fully evaluating $F(J^{(t)})$, by using techniques from incremental view maintenance. We show in this section that the semi-naïve algorithm can be generalized from datalog to datalog$^\circ$, for certain restricted POPS, called complete, distributive dioids.

A *dioid* is a semiring $\boldsymbol{S} = (S, \oplus, \otimes, 0, 1)$ for which $\oplus$ is idempotent. Dioids have many applications in a wide range of areas, see [14] for an extensive coverage. Any diod is naturally ordered, and the natural order $a \preceq_S b$ holds iff $a \oplus b = b$.

Definition 6.1. *A POPS $\boldsymbol{S} = (S, \oplus, \otimes, 0, 1, \sqsubseteq)$ is called a* complete, distributive dioid *if $(S, \oplus, \otimes, 0, 1)$ is a dioid, $\sqsubseteq$ is the dioid's natural order, and the ordered set $(S, \sqsubseteq)$ is a* complete, distributive lattice*, which means that every set $A \subseteq S$ has a greatest lower bound $\bigwedge A$, and $x \vee \bigwedge A = \bigwedge\{x \vee a \mid a \in A\}$. In a complete, distributive dioid, the* difference *operator is defined by*

$$b \ominus a \stackrel{def}{=} \bigwedge\{c \mid a \oplus c \sqsupseteq b\} \tag{10}$$

There are many examples of complete, distributive dioids: $(2^{\boldsymbol{U}}, \cup, \cap, \emptyset, \boldsymbol{U}, \subseteq)$ is a complete, distributive dioid, whose difference operator is exactly the set-difference $b - a = \bigcap\{c \mid b \subseteq a \cup c\} = b \setminus a$. The tropical semiring Trop^+ is also a complete, distributive dioid, whose difference operator is shown in Eq. (4) of the Introduction. On the other hand, Trop^+_p, $\mathsf{Trop}^+_{\leq\eta}$, $\mathbb{R}_\perp$ are not dioids.

We prove in the full version [15]:

Theorem 6.2. *Consider a datalog$^\circ$ program over a complete, distributive dioid. Then the Semi-naïve Algorithm 2 returns the same answer as the Naïve Algorithm 1.*

The main advantage of the semi-naïve algorithm comes from the fact that the difference $F(I, J^{(t)}) \ominus J^{(t)}$ can be computed incrementally. To see this, assume for simplicity that we have a single IDB predicate T and a single EDB predicate E: the datalog$^\circ$ program consists of a single rule, T :- $F(E, T)$. Recall that F is a sum-sum-product expression, and we will write it as $F(E, T) = G_0(E) \oplus G(E, T)$, where G_0 does not depend on T, and each sum-product expression in G contains at least one occurrence of T. Then, the reader may check that the following hold: $\delta^{(0)} = G_0(E)$ and $\delta^{(t)} = G(E, T^{(t)}) \ominus T^{(t)} = G(E, T^{(t-1)} \oplus \delta^{(t-1)}) \ominus T^{(t)}$ for $t \geq 1$. If G is linear in T (i.e. each sum-product contains exactly one occurrence of T) then $\delta^{(t)} = G(E, \delta^{(t-1)}) \ominus T^{(t)}$ for $t \geq 1$, When G is non-linear, let k be an upper bound on the number of occurrences of T in any sum-product expression of G. Then we can give new names to the different occurrences of T and write $G(E, T) = H(E, T, T, \ldots, T)$, where $H(E, T_1, \ldots, T_k)$ is a sum-sum-product expression where each sum-product contains at most one occurrence of T_i, for $i = 1, k$. There are now two ways to compute the expression $G(E, T^{(t-1)} \oplus \delta^{(t-1)})$ incrementally. One is to expand it into a sum of $2^k - 1$ terms, where each argument T_i is replace by either $T^{(t-1)}$ or $\delta^{(t-1)}$, with at least one argument being replaced by $\delta^{(t-1)}$. An alternative is to use the following *differential rule*, which is a sum of only t terms:

$$\delta^{(t)} \leftarrow \left(\bigoplus_i H(E, \underbrace{T^{(t)}, \ldots, T^{(t)}, \delta^{(t-1)}, T^{(t-1)}, \ldots, T^{(t-1)}}_{\delta^{(t-1)} \text{ is on position } i}) \right) \ominus T^{(t)}$$

7. PRACTICAL APPLICATIONS AND CONSIDERATIONS

This section outlines some of the main applications and further challenges we have to resolve when we apply datalog$^\circ$ in a practical database management system.

The first challenge is that the user-facing query language of relational systems is not datalog$^\circ$, and not even datalog. There were some attempts at exposing the semi-ring abstractions directly to the users [9]; however, it is not hard to see why widespread adoption and algebraic abstractions do not go hand in hand. On the other hand, datalog$^\circ$ and its algebraic variants [17] are *perfect* to be used as an intermediate representation in the query evaluation and optimization pipeline. (We use these Algebraic Intermediate Representations at RelationalAI.) Once we obtain the recursive queries written in datalog$^\circ$, optimization techniques described in this paper apply, and the runtime can "'take it from there".

The main challenge lies in the parsing step, where we have to map the queries written in some other front-end language such as SQL or Rel into datalog$^\circ$. Parsing is fundamentally an under-specified problem. For instance, some queries can both be mapped to the (max, +) semiring and the (max, ×) semiring. The domain of the semiring sometimes is required to be non-negative, which is not trivial to infer from the data / query.

The second challenge is in query optimization. Non-recursive query optimization, while remaining a very challenging problem, is a well-understood problem. Recursive query optimization is a completely different beast. At the start of the recursion there are no statistics on the IDBs for us to do cardinality estimation and query planning. Re-optimzing the query in the middle of execution is an option (as in Adaptive Query Processing), at the cost of query recompilation time and random access, leading to significant latency.

The third challenge is on how to implement some of the optimization techniques that look like a good idea on paper. Semi-naïve evaluation, for example, is a good idea on paper because every iteration makes use of only the "delta" from the previous iteration. However, depending on the query shape and the data size, it is far from always true that semi-naïve evaluation is faster. The differential rule described in the previous section means we have two choices for every occurrence of an IDB: we either keep for each IDB T_i two copies $T_i^{(t)}$ and $T_i^{(t-1)}$, or we keep one copy $T_i^{(t-1)}$ and evaluate the same rule an exponential number of times, depending on which combination of $T_i^{(t-1)}$ and $\delta T_i^{(t)}$ we use. If T was the transitive closure of a big graph, then keeping two copies of T is prohibitive. If the number of IDBs is relatively large, then evaluating a rule an exponential number of times is not desirable. All these are on top of the second challenge.

Last but not least, in many use-cases on large data, users do not need the entire output of a recursively defined IDB. For example, they may want to compute the all-pairs shortest paths problem only to compute the diameter of the graph (maximum over the shortest path lengths) afterwards. These are the "demands" on the output of a recursion that sometimes we can optimize away without evaluating the recursion in full, as we showed in [32]. However, users may compose queries on IDBs in arbitrary ways, leading to very interesting and challenging optimization issues: when do we apply our paper-optimization technique? when do we evaluate it in full? how do the mutual recursions interact when we compose IDBs defined by different recursions?

8. CONCLUSIONS

A massive number of application domains demand us to move beyond the confine of the Boolean world: from program analysis [6, 26], graph algorithms [4, 23, 22], provenance [13], formal language theory [20], to machine learning and linear algebra [27, 1]. Semiring and poset theory – of which POPS is an instance – is the natural bridge connecting the Boolean island to these applications.

The bridge helps enlarge the set of problems datalog$^\circ$ can express in a very natural way. The possibilities are endless. For example, amending datalog$^\circ$ with an interpreted function such as sigmoid will allow it to express typical neural network computations. Adding another semiring to the query language helps express rectilinear units in modern deep learning. At the same time, the bridge facilitates the porting of analytical ideas from datalog to analyze convergence properties of the application problems, and to carry over optimization techniques such as semi-naïve evaluation.

This paper established part of the bridge. There are many interesting open problems left; we mention a few here.

The question of whether a datalog$^\circ$ program over p-stable POPS converges in polynomial time in p and N is open.

In the functional aggregate queries [17] sense

This is open even for linear programs. Our result on Trop_p indicates that the linear case is likely in PTIME. If p-stability does not hold, then ACC was the next best barrier. It would be interesting to have a sufficient condition for convergence beyond ACC.

We can introduce negation to datalog$^\circ$ as an interpreted predicate. The question is, can we extend semantics results (such as stable model semantics) from general datalog / logic programming to datalog$^\circ$ with negation? The full version of this paper [15] contains more detailed discussions on this front. Beyond exact solution and finite convergence, as mentioned in the introduction, it is natural in some domain applications to have approximate fixpoint solutions, which will allow us to trade off convergence time and solution quality. A theoretical framework along this line will go a long way towards making datalog$^\circ$ deal with real machine learning, linear algebra, and optimization problems.

9. REFERENCES

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P. A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016* (2016), K. Keeton and T. Roscoe, Eds., USENIX Association, pp. 265–283.

[2] Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases.* Addison-Wesley, 1995.

[3] Aji, S. M., and McEliece, R. J. The generalized distributive law. *IEEE Trans. Inf. Theory 46*, 2 (2000), 325–343.

[4] Carré, B. *Graphs and networks.* The Clarendon Press, Oxford University Press, New York, 1979. Oxford Applied Mathematics and Computing Science Series.

[5] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to algorithms*, third ed. MIT Press, Cambridge, MA, 2009.

[6] Cousot, P., and Cousot, R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977* (1977), R. M. Graham, M. A. Harrison, and R. Sethi, Eds., ACM, pp. 238–252.

[7] Dannert, K. M., Grädel, E., Naaf, M., and Tannen, V. Semiring provenance for fixed-point logic. In *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)* (2021), C. Baier and J. Goubault-Larrecq, Eds., vol. 183 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 17:1–17:22.

[8] Dechter, R. Bucket elimination: a unifying framework for processing hard and soft constraints. *Constraints An Int. J. 2*, 1 (1997), 51–55.

[9] Eisner, J., and Filardo, N. W. Dyna: Extending datalog for modern AI. In *Datalog Reloaded - First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers* (2010),

O. de Moor, G. Gottlob, T. Furche, and A. J. Sellers, Eds., vol. 6702 of *Lecture Notes in Computer Science*, Springer, pp. 181–220.

[10] Fitting, M. A kripke-kleene semantics for logic programs. *J. Log. Program. 2*, 4 (1985), 295–312.

[11] Freeman, L. C. A Set of Measures of Centrality Based on Betweenness. *Sociometry 40*, 1 (Mar. 1977), 35–41.

[12] Gondran, M., and Minoux, M. *Graphs, dioids and semirings*, vol. 41 of *Operations Research/Computer Science Interfaces Series*. Springer, New York, 2008. New models and algorithms.

[13] Green, T. J., Karvounarakis, G., and Tannen, V. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China* (2007), L. Libkin, Ed., ACM, pp. 31–40.

[14] Gunawardena, J. An introduction to idempotency. In *Idempotency (Bristol, 1994)*, vol. 11 of *Publ. Newton Inst.* Cambridge Univ. Press, Cambridge, 1998, pp. 1–49.

[15] Khamis, M. A., Ngo, H. Q., Pichler, R., Suciu, D., and Wang, Y. R. Convergence of datalog over (pre-) semirings. *CoRR abs/2105.14435* (2021).

[16] Khamis, M. A., Ngo, H. Q., Pichler, R., Suciu, D., and Wang, Y. R. Convergence of datalog over (pre-) semirings. In *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022* (2022), L. Libkin and P. Barceló, Eds., ACM, pp. 105–117.

[17] Khamis, M. A., Ngo, H. Q., and Rudra, A. 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* (2016), T. Milo and W. Tan, Eds., ACM, pp. 13–28.

[18] Kohlas, J. *Information algebras - generic structures for inference.* Discrete mathematics and theoretical computer science. Springer, 2003.

[19] Kohlas, J., and Wilson, N. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artif. Intell. 172*, 11 (2008), 1360–1399.

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

[21] Lehmann, D. J. Algebraic structures for transitive closure. *Theor. Comput. Sci. 4*, 1 (1977), 59–76.

[22] Lipton, R. J., Rose, D. J., and Tarjan, R. E. Generalized nested dissection. *SIAM J. Numer. Anal. 16*, 2 (1979), 346–358.

[23] Lipton, R. J., and Tarjan, R. E. Applications of a planar separator theorem. *SIAM J. Comput. 9*, 3 (1980), 615–627.

[24] Liu, Y. A., and Stoller, S. D. Founded semantics and constraint semantics of logic rules. *J. Log. Comput. 30*, 8 (2020), 1609–1668.

[25] Liu, Y. A., and Stoller, S. D. Recursive rules with aggregation: A simple unified semantics, 2020.

[26] Nielson, F., Nielson, H. R., and Hankin, C. *Principles of program analysis.* Springer-Verlag, Berlin, 1999.

[27] Rocktäschel, T. Einsum is all you need - Einstein summation in deep learning. `https://rockt.github.io/2018/04/30/einsum`.

[28] Rote, G. Path problems in graphs. In *Computational graph theory*, vol. 7 of *Comput. Suppl.* Springer, Vienna, 1990, pp. 155–189.

[29] Scott, D., and Strachey, C. Toward a mathematical semantics for computer languages, 1971.

[30] Shenoy, P. P., and Shafer, G. Axioms for probability and belief-function proagation. In *UAI '88: Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence, Minneapolis, MN, USA, July 10-12, 1988* (1988), R. D. Shachter, T. S. Levitt, L. N. Kanal, and J. F. Lemmer, Eds., North-Holland, pp. 169–198.

[31] Vardi, M. Y. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA* (1982), H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber, Eds., ACM, pp. 137–146.

[32] Wang, Y. R., Khamis, M. A., Ngo, H. Q., Pichler, R., and Suciu, D. Optimizing recursive queries with progam synthesis. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022* (2022), Z. Ives, A. Bonifati, and A. E. Abbadi, Eds., ACM, pp. 79–93.

[33] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. Apache spark: a unified engine for big data processing. *Commun. ACM 59*, 11 (2016), 56–65.

[34] Zaniolo, C., Yang, M., Interlandi, M., Das, A., Shkapsky, A., and Condie, T. Declarative bigdata algorithms via aggregates and relational database dependencies. In *Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, May 21-25, 2018* (2018), D. Olteanu and B. Poblete, Eds., vol. 2100 of *CEUR Workshop Proceedings*, CEUR-WS.org.