A Back-to-basics empirical study of Datalog

BRUNO RUCY CARNEIRO ALVES DE LIMA[*] and G.K.M. TOBIN[*], Institute for Clarity in Documentation, USA
LARS THØRVÄLD, The Thørväld Group, Iceland
VALERIE BÉRANGER, Inria Paris-Rocquencourt, France
APARNA PATEL, Rajiv Gandhi University, India
HUIFEN CHAN, Tsinghua University, China
CHARLES PALMER, Palmer Research Laboratories, USA
JOHN SMITH, The Thørväld Group, Iceland
JULIUS P. KUMQUAT, The Kumquat Consortium, USA

Fig. 1. Seattle Mariners at Spring Training, 2010.

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Additional Key Words and Phrases: datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Bruno Rucy Carneiro Alves de Lima, G.K.M. Tobin, Lars Thørväld, Valerie Béranger, Aparna Patel, Huifen Chan, Charles Palmer, John Smith, and Julius P. Kumquat. 2018. A Back-to-basics empirical study of Datalog. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 19 pages. https://doi.org/XXXXXXX.XXXXXXX

[*]Both authors contributed equally to this research.

1 INTRODUCTION

Motivation. SQL has been the *de facto* universal relational database interface for querying and management since its inception, with all other alternatives having had experienced disproportionately little interest. The reasons for this are many, out of which only one is relevant to this narrative; performance. Curiously, there doesn't seem to exist a seminal article that investigates this event either from the technological or antropological viewpoint.

The runner-ups of popularity were languages used for machine reasoning, a subset of the Artificial Intelligence field that attempts to attain intelligence through the usage of rules over knowledge. The canonical language for reasoning over relational databases is datalog[**?**]. Similarly to SQL, it also is declarative, however, its main semantics' difference is in the support for recursion while still ensuring termination irrespective of the program being run.

A notable issue with respect to real-world adoption of datalog is in tractability. The combined complexity of evaluating a program is EXPTIME[**?**], while SQL queries are AC^0. It was not until recently[**? ?**] that scalable implementations were developed.

Digital analytics has been one of the main drivers of the recent datalog renaissance, with virtually all big-data oriented datalog implementations having had either been built on top of the most mainstream industry oriented frameworks[**? ? ?**] or with the aid of the most high-profile technology companies[**? ? ?**].

Another strong source of research interest has been from the knowledge graph community. A knowledge graph *KG* is a regular relational database *I* that contains *ground truths*, and rules. The most important operation is called *materialization*, the derivation of all truths that follow from the application of rules over the relational database, with the most straightforward goal being to ensure queries to have the lowest latency.

Problem. Seeking ways to introduce tuple-generating dependencies to datalog programs, with evaluation remaining tractable, has been one of the most active research directions, with highly-influential papers establishing new families of datalog languages[**?**] and thoroughly exploring their complexity classes alongside further expansions[**? ? ?**].

These advancements have been somewhat tested in practice, albeit with no full reference implementation having been specified. The most comprehensive, and recent, is closed-source[**?**]. The leading datalog engine in general, is also closed-source[**?**], with no open-source implementation having had attained any level of popularity, despite the relative simplicity of the language itself.

The two most popular datalog-related projects are DataScript[**?**] and Open Policy Agent[**?**], with the former being a top-down engine whose novelty lies in covering much functionality from a proprietary project, Datomic[**?**], while being implemented on top of a simple in-memory B-Tree. The latter is also a top-down evaluator, with severely limited usage of recursion. Neither of these projects have an intrinsic didactic nor scientific value.

The lack of a canonical open-source implementation of datalog makes attempts at making empirical statements about performance-impacting theoretical developments brittle and difficult, since there is no point of reference to compare and validate, and comparisons against commercial implementations are not reliable, since optimizations might be trade secrets.

A notorious exploration that highlights this issue is the COST, Configuration That Outperforms a Single Thread, article[**?**], in which the author posits that multiple published graph-processing engines are likely never able to outperform simple single-threaded implementations. Some high-profile datalog implementations were built upon systems mentioned on that article. Later on, the author made multiple pieces of informal writing in which the most performant datalog engines were investigated for COST[**? ?**], with results that showed a very different picture than the ones depicted by the original article's.

Methodology. To address the aforementioned problem, we conduct a back-to-basics empirical study of datalog evaluation, revisiting and measuring the core assumptions that go into the implementation of an evaluator. Straightforward single-threaded, parallel and distributed implementations of both substitution-based and relational-algebra interpretations are realized alongside common well-known optimizations. Due to the popularity of the relational approach, we give special focus to the problem of choosing an indexing data structure, and investigate several alternatives, including a novel one, to the ubiquitous BTree.

Contributions. In this article we make several contributions to clarifying, benchmarking and easing pushing the boundaries of datalog evaluation engineering research further by providing performant and open-source implementations of single-threaded, parallel and distributed evaluators.

- **Techniques and Guidelines.** We study the challenge of building a reasoner from scratch, with no underlying framework, and ponder over all decisions necessary in order to materialize that, alongside with relevant recent literature.
- **New Data Structure.** We introduce the Spine, a simple and clever alternative to the B-Tree that exhibits competitive performance in all benchmarked datalog workloads.
- **Implementation.** All code outputs of this article are coalesced in a rust library named shapiro, consisted of a datalog to relational algebra rewriter, a relational algebra evaluator, and two datalog engines, one that is parallel-capable and supports both substitution-based and relational algebra methods, and other that relies on the state-of-the-art differential dataflow[?] distribution computation framework. The main expected outcome of this library is to provide well-understood-and-reasoned-about baseline implementations from where future research can take advantage of, and reliable COST configurations can be attained.
- **Benchmarking.** We perform two thorough benchmark suites. One evaluates the performance of the developed relational reasoner with multiple different index data structures, and another that compares the performance of the distributed reasoner against four state-of-the-art distributed reasoners. The selected datasets are either from the program analysis, heavy users of not-distributed datalog, or from the semantic web community , which has multiple popular infinitely-scalable benchmarks, and are the main proponents of existential datalog.

2 RELATED WORK

Datalog engines. There are two kinds of recent relevant datalog engines. The first encompasses those that push the performance boundary, with the biggest proponents being RDFox[?], that proposes the to-date, according to their benchmarks, most scalable parallelisation routine, RecStep[?], that builds on top of a highly efficient relational engine, and DCDatalog[?], that builds upon an influential query optimizer, DeALS[?] and extends a work that establishes how some linear datalog programs could be evaluated in a lock-free manner, to general positive programs.

One of the most high-profile datalog papers of interest has been BigDatalog[?], that originally used the query optimizer DeALs, and was built on top of the very popular Spark[?] distribution framework. Soon after, a prototypical implementation[?] over Flink[?], a distribution framework that supports streaming, Cog, followed. Flink, unlike Spark, supports iteration, so implementing reasoning did not need to extend the core of the underlying framework. The most successful attempt at creating a distributed implemention has been Nexus[?], that is also built on Flink, and makes use of its most advanced feature, incremental stream processing. To date, it is the fastest distributed implementation.

Data structures used in datalog engines. The core of each datalog engine is consisted of possibly two main data structures: one to hold the data itself, and another for indexes. Surprisingly little regard is given to this, compared to algorithms themselves, despite potentially being one of the most detrimental factors for performance . Binary-decision diagrams[**?**], hash sets[**?**] and B-Trees[**?**] are often used as either one or both main structures. An important highlight of the importance of data structure implementation is how in [**?**] Subotic et al, managed to attain an almost 50 times higher performance in certain benchmarks than other implementations of the same data structure.

3 DATALOG EVALUATION

In this section we review the basics of all concepts related to datalog evaluation, as it is done in the current time.

3.1 Datalog

Datalog[**?**] is a declarative programming language. A program P is a set of rules r, with each r being a restricted first-order formula of the following form:

$$\bigwedge_{i=1}^{k} B_i(x_1, ..., x_j) \rightarrow \exists(y_1, ..., y_j)H(x_1, ..., x_j, y_1, ..., y_j)$$

with k, j as finite integers, x and y as terms, and each B_i and H as predicates. A term can belong either to the set of variables, or constants, however, it is to be noted that all y are existentially quantified. The set of all B_i is called the *body*, and H the *head*.

A rule r is said to be datalog, if the set of all y is empty, and no predicate is negated, conversely, a datalog program is one in which all rules are datalog.

Example 3.1. Datalog Program

$$P = \left\{ \text{SubClassOf}(?x, ?y) \wedge \text{SubClassOf}(?y, ?z) \rightarrow \text{SubClassOf}(?x, ?z) \right\}$$

Example 3.1 shows a simple valid recursive program. The only rule denotes that *for all x, y, z, if x is in a SubClassOf relation with y, and y is in a SubClassOf relation with z, then it follows that x is in a subClassOf relation with z.*

The meaning of a datalog program is often[**?**] defined through a *Herbrand Interpretation*. The first step to attain it is the *Herbrand Universe* $\mathfrak{U}$, the set of all constant, commonly referred to as *ground*, terms.

Example 3.2. Herbrand Universe

$$S = \left\{ \begin{array}{l} \text{SubClassOf}(\text{professor}, \text{employee}) \\ \text{SubClassOf}(\text{employee}, \text{taxPayer}) \\ \text{SubClassOf}(\text{employee}, \text{employed}) \\ \text{SubClassOf}(\text{employed}, \text{employee}) \end{array} \right\}$$

$$\mathfrak{U} = \left\{ \text{professor}, \text{employee}, \text{employed}, \text{taxPayer} \right\}$$

From the shown universe on example 3.2, it is possible to build The *Herbrand Base*, the set of all possible truths, from *facts*, assertions that are true, as represented by the actual constituents of the SubClassOf set.

Example 3.3. Herbrand Base

$$\mathfrak{B} = S \cup \left\{ \begin{array}{l} \text{SubClassOf(professor, } professor) \\ \text{SubClassOf(employee, } employee) \\ \text{SubClassOf(employed, } employed) \\ \text{SubClassOf(taxPayer, } taxPayer) \\ \text{SubClassOf(professor, } taxPayer) \\ \text{SubClassOf(taxPayer, } professor) \\ \text{SubClassOf(employee, } professor) \\ \text{SubClassOf(taxPayer, } employee) \end{array} \right\}$$

On example 3.3, all facts are indeed *possible*, but not necessarily *derivable* from the actual data and program. An interpretation I is a subset of $\mathfrak{B}$, and a *model* is an interpretation such that all rules are satisfied. A rule is satisfied if either the head is true, or if the body is not true.

Example 3.4. Models

$$I_1 = S \cup \left\{ \begin{array}{l} \text{SubClassOf(professor, } taxPayer) \\ \text{SubClassOf(employee, } employee) \end{array} \right\}$$

$$I_2 = S \cup \left\{ \begin{array}{l} \text{SubClassOf(professor, } taxPayer) \\ \text{SubClassOf(employee, } employee) \\ \text{SubClassOf(employed, } employed) \end{array} \right\}$$

$$I_3 = S \cup \left\{ \begin{array}{l} \text{SubClassOf(professor, } taxPayer) \\ \text{SubClassOf(employee, } employee) \\ \text{SubClassOf(employed, } employed) \\ \text{SubClassOf(professor, professor)} \end{array} \right\}$$

The first interpretation, I_1, from example 3.4, is not a model, since SubClassOf(employed, employed) is satisfied and present. Despite both I_2 and I_3 being models, I_2 is the *minimal* model, which is the definition of the meaning of the program over the data. The input data, the database, is named as the *Extensional Database EDB*, and the output of the program is the *Intensional Database IDB*.

Let an $DB = EDB \cup IDB$, and for there to be a program P. We define the *immediate consequence* of P over DB as all facts that are either in DB, or stem from the result of applying the rules in P to DB. The *immediate consequence operator* $\mathbf{I}_C(DB)$ is the union of DB and its immediate consequence, and the IDB, at the moment of the application of $\mathbf{I}_C(DB)$ is the difference of the union of all previous DB with the EDB.

It is trivial to see that $I_C(DB)$ is monotone, and given that both the EDB and P are finite sets, and that $IDB = \emptyset$ at the start, at some point $I_C(DB) = DB$, since there won't be new facts to be inferred. This point is the *least fixed point* of $I_c(DB)$[**?**], and happens to be the *minimal* model.

Example 3.5. Repeated application of I_c

$$
\begin{aligned}
P = \{Edge(?x, ?y) \rightarrow TC(?x, ?y), TC(?x, ?y), TC(?y, ?z) \rightarrow TC(?x, ?z)\} \\
EDB = \{Edge(1, 2), Edge(2, 3), Edge(3, 4)\} \\
DB = EDB \\
DB = I_C(DB) \\
DB == EDB \cup \{TC(1, 2), TC(2, 3), TC(3, 4)\} \\
DB = I_C(DB) \\
DB == EDB \cup \{TC(1, 2), TC(2, 3), TC(3, 4), TC(1, 3), TC(2, 4)\} \\
DB = I_C(DB) \\
DB == EDB \cup \{TC(1, 2), TC(2, 3), TC(3, 4), TC(1, 3), TC(2, 4), TC(1, 4)\} \\
DB = I_C(DB) \\
DB == EDB \cup \{TC(1, 2), TC(2, 3), TC(3, 4), TC(1, 3), TC(2, 4), TC(1, 4)\} \\
IDB = DB \setminus EDB
\end{aligned}
$$

The introduced form of evaluation, with a walkthrough given on example 3.5, is called *naive*, meanwhile, the ubiquitous evaluation mechanism, as of the date of writing this paper, is the *semi-naive* one. The only difference is that *semi-naive* does not repeatedly union the EDB with the entire IDB, but does so only with the difference of the previous immediate consequence with the IDB. This can be hinted from the example, where each next application of I_c only renders new facts from the previous newly derived ones.

3.2 Infer

The most relevant performance-oriented aspect of both of the introduced evaluation mechanisms is the implementation of I_c itself. The two most high-profile methods to do so are either purely evaluating the rules, or rewriting them in some other imperative formalism, and executing it.

The Infer[**?**] algorithm is the simplest example of the former, and relies on substitutions. A substitution S is a homomorphism $[x_1 \rightarrow y_1, ..., x_i \rightarrow y_i]$, such that x_i is a variable, and y_i is a constant. Given a not-ground fact, such as $TC(?x, 4)$, *applying* the substitution $[?x \rightarrow 1]$ to it will yield the ground fact $TC(1, 4)$.

Infer relies on attempting to build and extend substitutions for each fact in each rule body over every single DB fact. Once all substitutions are made, they are applied to the heads of each rule. Every result of this application that is ground belongs to the immediate consequence.

3.3 Relational Algebra

Relational Algebra[**?**] is an imperative language, that explicitly denotes operations over sets of tuples with fixed arity, relations. It is the most popular database formalism that there is, with virtually every single major database system adhering to the relational model[**? ? ?**] and supporting relational algebra as the SQL compilation target.

Let R and T be relations with arity r and t, θ be a binary operation with a boolean output, $R(i)$ be the i-th column in R, and $R[h, ..., k]$ be the subset of R such that only the columns $h, ..., k$ remain, and Const the set of all constant terms. The following are the most relevant relational algebra operators and their semantics:

- Selection by column $\sigma_{i=j)}(R) = \{a \in R | a(i) == a(j)\}$
- Selection by value $\sigma_{i=k}(R) = \{a \in R | a(i) == k\}$
- Projection $\pi_{h,...,k}(R) = \{(R(i), ..., R(j), \overrightarrow{C}) | i, j >= 1 \land i, j <= r \ \land \forall c \in C.c \in \text{Const}$
- Product $\times(R, T) = \{(a, b) | a \in R \land b \in T\}$
- Join $\bowtie_{i=j} = \{(a, b) | a \in R \land b \in T \land a(i) == b(j)\}$

Rewriting datalog into some form of relational algebra has been the most successful strategy employed by the vast majority of all current state-of-the-art reasoners[**? ? ? ? ? ?**] mostly due to the extensive industrial and academic research into developing data processing frameworks that process very large amounts of data, and the techniques that have arisen from these.

In spite of this, there is no open-source library that provides a stand-alone datalog to relational algebra translator, therefore every single datalog evaluator has to repeat this effort. Moreover, datalog rules translate to a specific form of relational algebra expressions, the select-project-join $\mathcal{SPJ}$ form.

A relational algebra expression is in the $\mathcal{SPJ}$ form if it consists solely of select, project and join operators. This form is very often seen in practice, being equivalent to `SELECT ... FROM ... WHERE ...` SQL queries, and highly benefits from being equivalent to conjunctive queries, that are equivalent to single-rule and non-recursive datalog programs.

We propose a straightforward not-recursive pseudocode algorithm to translate a datalog rule into a SPJ expression tree, in which relational operators are nodes and leaves are relations. The value proposition of the algorithm is for the resulting tree to be ready to be recursively executed, alongside having two essential relational optimizations, selection by value pushdown, and melding selection by column with products into joins, the most important relational operator. The canonical algorithms for translating datalog to relational algebra[**? ?**] are recursive, complex, and do not assume the output to be a tree, instead being mostly symbolic.

4 SHAPIRO

In order to coalesce all contributions in this paper, an extensible reasoning system was designed, from scratch. No frameworks were used, nor any code related to any other reasoner has been reutilized, furthermore, a modern, performant and safe programming language was used, Rust[**?**].

The main rationale for this choice is twofold: garbage-collected languages such as java are hard to benchmark, and tuning performance, alongside reasoning about it, often requires the developer to either learn highly specific minutiae relating to the compiler or interpreter, and, or, the garbage collector.

All most performant shared-memory reasoners are unsurpisingly all implemented in C++[**? ? ?**], since it provides manual memory management and is the *de facto* language for high-performance computing. Rust is a recent language, approximately 10 years old, that exhibits similar or faster performance than C++, with its *raison d'etre* being that, unlike C++, it is memory-safe and thread-safe[**? ? ?**]; both of these guarantees are of incredible immediate benefit to writing a reasoner.

Shapiro, the developed system, is fully modular and offers a heap of independent components and interfaces.

On figure 2 we can see the most relevant modules. The northwestern quadrant, Traits, refers to Rust Traits, constructs that define *behavior*, and that can be passed to functions. It is important to take note that Rust favors *composition* over inheritance, unlike Java. Nevertheless, all other quadrants are built upon these traits, and are therefore generic over any type that implements them.

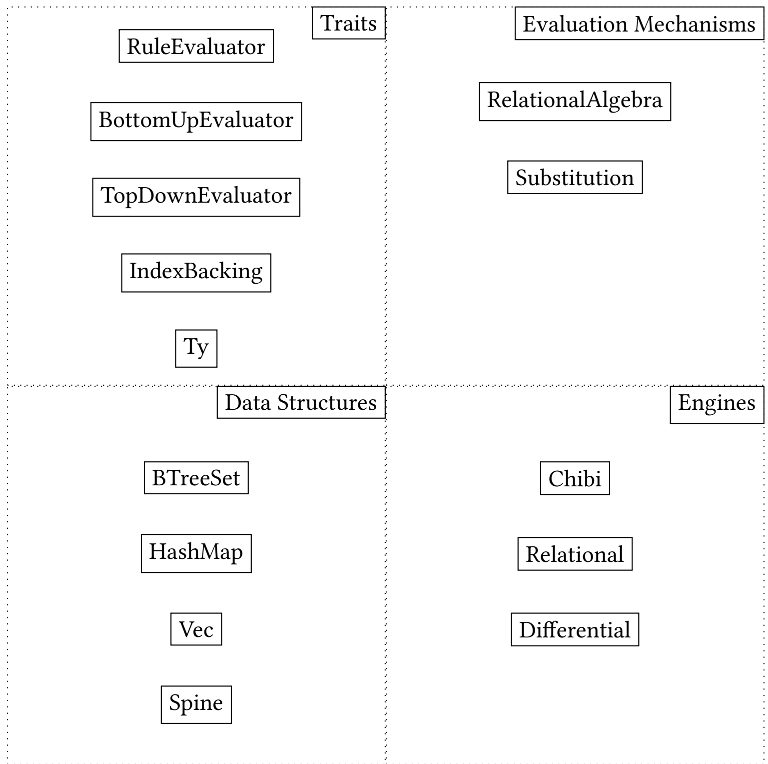

Fig. 2. Shapiro Components

A `Rule Evaluator` represents $\mathbf{T}_p$, requiring only that types implement a function `evaluate`, that accepts an instance, and produces another instance. The evaluation mechanisms `Relational Algebra` and `Substitution` both implement this trait, evaluating datalog rules by either using the aforegiven translation algorithm, or using Infer. We also provide naive parallel evaluation of rules, in which each rule, in both mechanisms, are sent to be executed in a threadpool. We leverage the highly efficient, and easy to use, requiring only one additional line of code, rayon[**?**] library. This should suffice as a transparent baseline from which other parallelization strategies could be compared to.

The concept of fixpoint evaluation is a Struct, that consists of the ground fact database, *EDB*, a `Rule Evaluator`, and an array of instances. It is assumed that the datalog program is in the instance itself. Semi-naive evaluation is a method of fixpoint evaluation that takes no arguments, and instead uses two additional instances, one to keep the current delta, and the previous one. This is succintly implemented in rust in the same manner as this description.

`BottomUpEvaluator` and `TopDownEvaluator` both respectively denote the two kinds of evaluation, respectively, the explicit materialization of the program, and the answering to a query with respect to a program. The former requires the implementation of a function that takes in only a program as an argument, and the latter, a program and a goal.

The three engines, `Chibi`, `Relational` and `Differential` all implement only `BottomUpEvaluator`. In order to be able to evaluate a program, it is required to have access to an instance. An instance is defined in code as it is in literature, a set of relations. Taking inspiration from the highly successful open-source project DataScript[**?**], we implement relations as hashmaps, with generic hashing functions. The point of using a hashmap is that it allows for one to easily disregard duplicates altogether, further simplifying the implementation.

Indexes on the other hand, are a vital optimization aspect. A considerable amount of recent datalog research has dealt with speeding up relational joins with respect to datalog, such as in attempting to specialize data structures to it[?], and rminimizing the number of necessary joins to evaluate a SPJ expression[?]. Thus, a trait `IndexBacking` is defined, whose requirements are two methods, one that takes in a row for insertion, and another that requires a join implementation.

We implement the `IndexBacking` trait for implementations of all most used data structures for datalog indexes, such as the Rust standard library BTree[?] and HashMap, persistent implementations of both them, a regular vector, that naively sorts itself before a join, and the Spine, an experimental data structure, that empirically shows good performance characteristics in datalog workflows.

4.1 The Spine

The Spine is a simple two-level pointer-free data structure comprised of arrays, and an index. Sorted Arrays are the fastest data structure for binary search lookups. Compared to search trees, they are much more cache efficient, since every single piece of data is in one contiguous region in memory, and no pointer indirection is needed in every binary search iteration.

Insertion in sorted arrays is commonly implemented throgh an emulated binary insertion sort, where the position of the to-be-inserted element is first found through a binary search, and then all elements to the right of its supposed position are shifted. The complexity of this operation is $O(n)$, therefore being exponentially slower than search trees, rendering it unusuable in dynamic cases, such as in datalog evaluation, in which possibly costly bulk insertions occur at a very fast pace.

In modern CPU architectures, there are multiple levels of memory, going from CPU registers, L-caches, up to disk. Accessing one element cached in low-level memory can be hundreds of thousands of times faster than doing so in the disk, therefore it is possible, in practice, for asymptotic linear-time access to be faster than logarithmic.

The Spine is a naive attempt to take advantage of that, by keeping a totally ordered set of fixed-capacity B sorted arrays, ordered by their maximum, with B being determined empirically to be the value such that the performance ratio of insertion and search is as desired. Moreover, given that the most efficient data structure for datalog indexing, BTree, also relies on sorted data, it often occurs that operations benefit from both spatial and temporal cache locality.

Souffle's datalog-tailored B-Tree[?] has a simple yet effective *hint* mechanism for speeding up searches and insertions, by taking advantage of locality. Whenever a new row is inserted, its traversal is kept as a *hint*, therefore if the next row to be inserted falls within the same *range* as the previous, the *range* itself can be searched, possibly entirely avoiding a traversal, and only searching in the child node. This same mechanism is implemented in the Spine, with the help of its index.

Fenwick Trees[?] are highly specialized data structures used for dynamic cumulative sum. Given a fixed-length array of counts, the fenwick tree allows for calculating the prefix sum in $O(\log_2 n)$ time, while also providing adjustments to counts in the same bounds. We leverage the [?] in order to maintain an index that will keep track of the size of each internal array. The cost of updating the fenwick tree is minimal, in the context of the Spine, taking $O(\log_2(B))$ time with $B \ll n$.

In order to ensure that all sorted sub-arrays have

4.2 Differential Dataflow

most of the distributed datalog engines are built on top of either graph-processing or map-reduce frameworks their semantics do not fit datalog properly like a glove

Differential dataflow however, does. It substantially improves semi-naive evaluation by automatically parallelizing it. Make a graphic example showing how

5 EXPERIMENTS

I actually did some of the experiments. results were promising, trust me :D

ACM's consolidated article template, introduced in 2017, provides a consistent LaTeX style for use across ACM publications, and incorporates accessibility and metadata-extraction functionality necessary for future Digital Library endeavors. Numerous ACM and SIG-specific LaTeX templates have been examined, and their unique features incorporated into this single new template.

If you are new to publishing with ACM, this document is a valuable guide to the process of preparing your work for publication. If you have published with ACM before, this document provides insight and instruction into more recent changes to the article template.

The "`acmart`" document class can be used to prepare articles for any ACM publication — conference or journal, and for any stage of publication, from review to final "camera-ready" copy, to the author's own version, with *very* few changes to the source.

6 TEMPLATE OVERVIEW

As noted in the introduction, the "`acmart`" document class can be used to prepare many different kinds of documentation — a double-blind initial submission of a full-length technical paper, a two-page SIGGRAPH Emerging Technologies abstract, a "camera-ready" journal article, a SIGCHI Extended Abstract, and more — all by selecting the appropriate *template style* and *template parameters*.

This document will explain the major features of the document class. For further information, the *LaTeX User's Guide* is available from https://www.acm.org/publications/proceedings-template.

6.1 Template Styles

The primary parameter given to the "`acmart`" document class is the *template style* which corresponds to the kind of publication or SIG publishing the work. This parameter is enclosed in square brackets and is a part of the `documentclass` command:

```
\documentclass[STYLE]{acmart}
```

Journals use one of three template styles. All but three ACM journals use the `acmsmall` template style:

- `acmsmall`: The default journal template style.
- `acmlarge`: Used by JOCCH and TAP.
- `acmtog`: Used by TOG.

The majority of conference proceedings documentation will use the `acmconf` template style.

- `acmconf`: The default proceedings template style.
- `sigchi`: Used for SIGCHI conference articles.

- `sigchi-a`: Used for SIGCHI "Extended Abstract" articles.
- `sigplan`: Used for SIGPLAN conference articles.

6.2 Template Parameters

In addition to specifying the *template style* to be used in formatting your work, there are a number of *template parameters* which modify some part of the applied template style. A complete list of these parameters can be found in the *LaTeX User's Guide.*

Frequently-used parameters, or combinations of parameters, include:

- `anonymous,review`: Suitable for a "double-blind" conference submission. Anonymizes the work and includes line numbers. Use with the `\acmSubmissionID` command to print the submission's unique ID on each page of the work.
- `authorversion`: Produces a version of the work suitable for posting by the author.
- `screen`: Produces colored hyperlinks.

This document uses the following string as the first command in the source file:

```
\documentclass[sigconf,authordraft]{acmart}
```

7 MODIFICATIONS

Modifying the template — including but not limited to: adjusting margins, typeface sizes, line spacing, paragraph and list definitions, and the use of the `\vspace` command to manually adjust the vertical spacing between elements of your work — is not allowed.

Your document will be returned to you for revision if modifications are discovered.

8 TYPEFACES

The "`acmart`" document class requires the use of the "Libertine" typeface family. Your TeX installation should include this set of packages. Please do not substitute other typefaces. The "`lmodern`" and "`ltimes`" packages should not be used, as they will override the built-in typeface families.

9 TITLE INFORMATION

The title of your work should use capital letters appropriately - https://capitalizemytitle.com/ has useful rules for capitalization. Use the `title` command to define the title of your work. If your work has a subtitle, define it with the `subtitle` command. Do not insert line breaks in your title.

If your title is lengthy, you must define a short version to be used in the page headers, to prevent overlapping text. The `title` command has a "short title" parameter:

```
\title[short title]{full title}
```

10 AUTHORS AND AFFILIATIONS

Each author must be defined separately for accurate metadata identification. Multiple authors may share one affiliation. Authors' names should not be abbreviated; use full first names wherever possible. Include authors' e-mail addresses whenever possible.

Grouping authors' names or e-mail addresses, or providing an "e-mail alias," as shown below, is not acceptable:

```
\author{Brooke Aster, David Mehldau}
\email{dave,judy,steve@university.edu}
\email{firstname.lastname@phillips.org}
```

The `authornote` and `authornotemark` commands allow a note to apply to multiple authors — for example, if the first two authors of an article contributed equally to the work.

If your author list is lengthy, you must define a shortened version of the list of authors to be used in the page headers, to prevent overlapping text. The following command should be placed just after the last `\author{}` definition:

```
\renewcommand{\shortauthors}{McCartney, et al.}
```

Omitting this command will force the use of a concatenated list of all of the authors' names, which may result in overlapping text in the page headers.

The article template's documentation, available at https://www.acm.org/publications/proceedings-template, has a complete explanation of these commands and tips for their effective use.

Note that authors' addresses are mandatory for journal articles.

11 RIGHTS INFORMATION

Authors of any work published by ACM will need to complete a rights form. Depending on the kind of work, and the rights management choice made by the author, this may be copyright transfer, permission, license, or an OA (open access) agreement.

Regardless of the rights management choice, the author will receive a copy of the completed rights form once it has been submitted. This form contains LaTeX commands that must be copied into the source document. When the document source is compiled, these commands and their parameters add formatted text to several areas of the final document:

- the "ACM Reference Format" text on the first page.
- the "rights management" text on the first page.
- the conference information in the page header(s).

Rights information is unique to the work; if you are preparing several works for an event, make sure to use the correct set of commands with each of the works.

The ACM Reference Format text is required for all articles over one page in length, and is optional for one-page articles (abstracts).

12 CCS CONCEPTS AND USER-DEFINED KEYWORDS

Two elements of the "acmart" document class provide powerful taxonomic tools for you to help readers find your work in an online search.

The ACM Computing Classification System — https://www.acm.org/publications/class-2012 — is a set of classifiers and concepts that describe the computing discipline. Authors can select entries from this classification system, via https://dl.acm.org/ccs/ccs.cfm, and generate the commands to be included in the LaTeX source.

User-defined keywords are a comma-separated list of words and phrases of the authors' choosing, providing a more flexible way of describing the research being presented.

CCS concepts and user-defined keywords are required for for all articles over two pages in length, and are optional for one- and two-page articles (or abstracts).

13 SECTIONING COMMANDS

Your work should use standard LaTeX sectioning commands: `section`, `subsection`, `subsubsection`, and `paragraph`. They should be numbered; do not remove the numbering from the commands.

Simulating a sectioning command by setting the first word or words of a paragraph in boldface or italicized text is **not allowed.**

14 TABLES

The "`acmart`" document class includes the "`booktabs`" package — https://ctan.org/pkg/booktabs — for preparing high-quality tables.

Table captions are placed *above* the table.

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material are found in the *LaTeX User's Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed output of this document.

Always use midrule to separate table header rows from data rows, and use it only for this purpose. This enables assistive technologies to recognise table headers and support their users in navigating tables more easily.

15 MATH EQUATIONS

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

15.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin . . . \end` construction or with the short form `$ . . . $`. You can use any of the symbols and structures, from α to ω, available in LaTeX [?]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\rightarrow\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

15.2 Display Equations

A numbered display equation—one set off by vertical space from the text and centered horizontally—is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\rightarrow\infty} x = 0 \tag{1}$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_{0}^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

16 FIGURES

The "`figure`" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below.

Your figures should contain a caption which describes the figure to the reader.

Figure captions are placed *below* the figure.

Every figure should also have a figure description unless it is purely decorative. These descriptions convey what's in the image to someone who cannot see it. They are also used by search engine crawlers for indexing images, and when images cannot be loaded.

A figure description must be unformatted plain text less than 2000 characters long (including spaces). **Figure descriptions should not repeat the figure caption – their purpose is to capture important information that is not already provided in the caption or the main text of the paper.** For figures that convey important and complex new information, a short text description may not be adequate. More complex alternative descriptions can be placed in an appendix and referenced in a short figure description. For example, provide a data table capturing the information in a bar chart, or a structured list representing a graph. For additional information regarding how best to write figure descriptions and why doing this is so important, please see https://www.acm.org/publications/taps/describing-figures/.

16.1 The "Teaser Figure"

A "teaser figure" is an image, or set of images in one figure, that are placed after all author and affiliation information, and before the body of the article, spanning the page. If you wish to have such a figure in your article, place the command immediately before the `\maketitle` command:

```
\begin{teaserfigure}
  \includegraphics[width=\textwidth]{sampleteaser}
  \caption{figure caption}
  \Description{figure description}
\end{teaserfigure}
```

Fig. 3. 1907 Franklin Model D roadster. Photograph by Harris & Ewing, Inc. [Public domain], via Wikimedia Commons. (https://goo.gl/VLCRBB).

17 CITATIONS AND BIBLIOGRAPHIES

The use of BibTeX for the preparation and formatting of one's references is strongly recommended. Authors' names should be complete — use full first names ("Donald E. Knuth") not initials ("D. E. Knuth") — and the salient identifying features of a reference should be included: title, year, volume, number, pages, article DOI, etc.

The bibliography is included in your source document with these two commands, placed just before the `\end{document}` command:

```
\bibliographystyle{ACM-Reference-Format}
\bibliography{bibfile}
```

where "`bibfile`" is the name, without the "`.bib`" suffix, of the BibTeX file.

Citations and references are numbered by default. A small number of ACM publications have citations and references formatted in the "author year" style; for these exceptions, please include this command in the **preamble** (before the command "`\begin{document}`") of your LaTeX source:

```
\citestyle{acmauthoryear}
```

Some examples. A paginated journal article [?], an enumerated journal article [?], a reference to an entire issue [?], a monograph (whole book) [?], a monograph/whole book in a series (see 2a in spec. document) [?], a divisible-book such as an anthology or compilation [?] followed by the same example, however we only output the series if the volume number is given [?] (so Editor00a's series should NOT be present since it has no vol. no.), a chapter in a divisible book [?], a chapter in a divisible book in a series [?], a multi-volume work as book [?], a couple of articles in a proceedings (of a conference, symposium, workshop for example) (paginated proceedings article) [? ?], a proceedings article with all possible elements [?], an example of an enumerated proceedings article [?], an informally published work [?], a couple of preprints [? ?], a doctoral dissertation [?], a master's thesis: [?], an online document / world wide web resource [? ? ?], a video game (Case 1) [?] and (Case 2) [?] and [?] and (Case 3) a patent [?], work accepted for publication [?], 'YYYYb'-test for prolific author [?] and [?]. Other cites might contain 'duplicate' DOI and URLs (some SIAM articles) [?]. Boris / Barbara Beeton: multi-volume works as books [?] and [?]. A couple of citations with DOIs: [? ?]. Online citations: [? ? ?]. Artifacts: [?] and [?].

18 ACKNOWLEDGMENTS

Identification of funding sources and other support, and thanks to individuals and groups that assisted in the research and the preparation of the work should be included in an acknowledgment section, which is placed just before the reference section in your document.

This section has a special environment:

```
\begin{acks}
...
\end{acks}
```

so that the information contained therein can be more easily collected during the article metadata extraction phase, and to ensure consistency in the spelling of the section heading.

Authors should not prepare this section as a numbered or unnumbered `\section`; please use the "`acks`" environment.

19 APPENDICES

If your work needs an appendix, add it before the "`\end{document}`" command at the conclusion of your source document.

Start the appendix with the "`appendix`" command:

```
\appendix
```

and note that in the appendix, sections are lettered, not numbered. This document has two appendices, demonstrating the section and subsection identification method.

20 MULTI-LANGUAGE PAPERS

Papers may be written in languages other than English or include titles, subtitles, keywords and abstracts in different languages (as a rule, a paper in a language other than English should include an English title and an English abstract). Use `language=...` for every language used in the paper. The last language indicated is the main language of the paper. For example, a French paper with additional titles and abstracts in English and German may start with the following command

```
\documentclass[sigconf, language=english, language=german,
               language=french]{acmart}
```

The title, subtitle, keywords and abstract will be typeset in the main language of the paper. The commands `\translatedXXX`, `XXX` begin title, subtitle and keywords, can be used to set these elements in the other languages. The environment `translatedabstract` is used to set the translation of the abstract. These commands and environment have a mandatory first argument: the language of the second argument. See `sample-sigconf-i13n.tex` file for examples of their usage.

21 SIGCHI EXTENDED ABSTRACTS

The "`sigchi-a`" template style (available only in LaTeX and not in Word) produces a landscape-orientation formatted article, with a wide left margin. Three environments are available for use with the "`sigchi-a`" template style, and produce formatted output in the margin:

- `sidebar`: Place formatted text in the margin.
- `marginfigure`: Place a figure in the margin.
- `margintable`: Place a table in the margin.

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

A RESEARCH METHODS

A.1 Part One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi malesuada, quam in pulvinar varius, metus nunc fermentum urna, id sollicitudin purus odio sit amet enim. Aliquam ullamcorper eu ipsum vel mollis. Curabitur quis dictum nisl. Phasellus vel semper risus, et lacinia dolor. Integer ultricies commodo sem nec semper.

A.2 Part Two

Etiam commodo feugiat nisl pulvinar pellentesque. Etiam auctor sodales ligula, non varius nibh pulvinar semper. Suspendisse nec lectus non ipsum convallis congue hendrerit vitae sapien. Donec at laoreet eros. Vivamus non purus placerat, scelerisque diam eu, cursus ante. Etiam aliquam tortor auctor efficitur mattis.

B ONLINE RESOURCES

Nam id fermentum dui. Suspendisse sagittis tortor a nulla mollis, in pulvinar ex pretium. Sed interdum orci quis metus euismod, et sagittis enim maximus. Vestibulum gravida massa ut felis suscipit congue. Quisque mattis elit a risus ultrices commodo venenatis eget dui. Etiam sagittis eleifend elementum.

Nam interdum magna at lectus dignissim, ac dignissim lorem rhoncus. Maecenas eu arcu ac neque placerat aliquam. Nunc pulvinar massa et mattis lacinia.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

Algorithm 1: An algorithm to translate a datalog rule into relational algebra

Input: A datalog rule $\mathcal{R}$
Result: A relational algebra expression $\mathcal{R}_a$

```
Function toIncompleteExpression(r : Datalog Rule) is
    let t be a fresh tree
    For For every fact a_i in the rule body b:
        create a relation node r_i with the same arity as a_i, and its terms representing columns. variable terms are column identifiers, and constant terms are
         temporary-lived proxies for selections. if i < len(b) − 1 then
            add a product node p_i to t. set r_i as the left child of p_i.
        else
            if len(b) > 1 then
                set r_i as the right child of p_{i−1}
        if i > 0 then
            set p_i as the right child of p_{i−1}
    return t
Function constantToSelection(e : Expression) is
    let t be a copy of e
    let C be a map C : Const → Var
    For every relation r_i in t:
        For every constant c_j in r_i:
            add a selection by value node s_j to t with column index j and value c_j
            set s_j's parent to r_i's, and r_i as its left child
            if ¬(c_j ∈ C) then
                create a fresh variable term v_j and store it in C with c_j as the key
            else
                replace c_j for the value in C under c_j
    return t
Function equalityToSelection(e : Expression) is
    let t be a copy of e
    let V be a map V : Var → ℤ
    let t_p be a pre-order traversal of t
    For every relation r_i in t_p:
        For every variable v_j in r_i:
            if ¬(v_j ∈ V) then
                add v_j to V with j as the value
            else
                let k be the value of v_j in V let p_i be the first product to the left of r_i in t_p
                add a selection by column node s_j to t with left column index k and right column index j
                set s_j's parent to p_i's, and p_i as its left child
    return t
Function projectHead(e : Expression, r : Datalog Rule) is
    let n be 0
    let t be a copy of e
    let h be the head of r
    let t_p be a pre-order traversal of t
    let V be a map V : Var → ℤ
    For every relation r_i in t_p:
        For every term x in r_i:
            if x is a variable then
                add x to V with n as value
            else
                continue
            n += 1
            add projection node z to t and set it as root with an empty list
            For every term x in h:
                if x is a constant then
                    push x into z
                else
                    let k be the value of x in V
                    push k into z
    return t
Function productToJoin(e : Expression) is
    let t be a copy of e
    let t_p be a pre-order traversal of t
    For every selection by column s_i in t_p:
        find the first product p_j after s_i in t_p
        remove s_i from t, swap p_j for a join g_j in t with left and right column indexes by those of s_i
    return t
R_a = toIncompleteExpression(R)
R_a = constantToSelection(expression, R)
R_a = equalityToSelection(expression, R)
R_a = projectHead(expression, R)
R_a = productToJoin(expression, R)
return R_a
```

Table 1. Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Table 2. Some Typical Commands

Command	A Number	Comments
`\author`	100	Author
`\table`	300	For tables
`\table*`	400	For wider tables