UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Merlin Kasesalu

Revisiting Query Magic

Master's Thesis (30 ECTS)

Supervisor:
Bruno Rucy Carneiro Alves de Lima, MSc

Tartu 2025

Revisiting Query Magic

Abstract:

Less research has gone into the practical aspects of Datalog relative to the theoretical ones. In this thesis, top-down evaluation and top-down-mimicking bottom-up evaluation (Magic Sets Transformation) are implemented and compared with regular bottom-up evaluation Datalog, using benchmarks on incrementally updated data. The findings demonstrate the advantage of Magic Sets Transformation in sparse graph structures with performance up to 2000 times faster than regular bottom-up evaluation, and of top-down evaluation in knowledge base reasoning tasks with performance up to 1000 times faster.

Keywords: Datalog, Magic Sets Transformation, Incremental Evaluation, Subsumptive Tabling

CERCS: P170 - Computer science, numerical analysis, systems, control

Pilk Maagiaga Päringutele

Lühikokkuvõte:

Datalogi praktilisi aspekte on teoreetilistega võrreldes vähem uuritud. Selles uurimistöös implementeeritakse ülalt-alla hindamine ja ülalt-alla jäljendavat alt-üles hindamine (*Magic Sets Transformation*) ning võrreldakse neid tavalise alt-üles hindamisega Datalogis inkrementaalselt uuendatavatel andmetel. Tulemused näitavad, et *Magic Sets Transformationi* kiirus hõredate graafistruktuuride puhul võib olla kuni 2000 korda kiirem kui tavalisel alt-üles hindamisel, ning ülalt-alla hindamisel kuni 1000 korda kiirem teadmusbaasi arutlusülesannetes.

Võtmesõnad: Datalog, Magic Sets Transformation, Inkrementaalne Hindamine, Subsumptive Tabling

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Table of Contents

1. Introduction

Modern data-driven applications increasingly deal with complex relationships that are naturally represented as graphs rather than traditional tabular structures. Social networks, knowledge graphs, biological networks, and organizational hierarchies all exhibit interconnected data where understanding relationships is as important as the data points themselves. However, traditional relational databases and SQL [1], while highly optimized for tabular data operations, face significant challenges when querying graph-like structures that require traversal.

Although modern SQL [2] implementations have introduced recursive common table expressions, these extensions remain limited to linear recursion and struggle with the complex, multi-path queries common in graph analysis. Tasks such as finding all nodes reachable from a given starting point, or analyzing hierarchical relationships often result in inefficient query plans in traditional SQL databases.

Datalog [3] is a declarative logic programming language based on Horn clauses, which naturally supports recursion and provides a framework for expressing complex graph queries. Where SQL requires complex self-joins and recursive constructs, Datalog expresses the same logic through simple, intuitive rules.

Less research has gone into the practical aspects of Datalog relative to the theoretical ones. If one wishes to create a new Datalog engine then they need to learn everything on their own. One of the goals of this thesis is to provide a reference for anyone starting out with Datalog, particularly with Magic Sets Transformation.

While most Datalog engines use bottom-up evaluation, top-down evaluation has also been shown to be an effective strategy in Datalog with even better performance than bottom-up in certain conditions [4].

The primary research question driving this work is: How do top-down evaluation and top-down-mimicking bottom-up evaluation (Magic Sets Transformation) compare to regular bottom-up evaluation Datalog?

In this thesis, the bottom-up evaluation optimization strategy, Magic Sets Transformation [5], and the top-down evaluation optimization strategy, Subsumptive Tabling [4], are implemented in the Datalog reasoner Micro-Datalog [6]. Their implementation is documented and benchmarks are run against pure semi-naïve evaluation of the same reasoner, and another Datalog reasoner, Ascent [7].

Most real-world data is dynamic and a useful reasoner should be able to make use of already inferred results when encountering new facts, instead of recomputing everything from scratch. On account of this reasoning, the benchmarks are run on incrementally updated data and the effect of different batch sizes on performance is investigated. Both synthetic and real-world data graphs are used as well as different query patterns.

The thesis makes both technical and empirical contributions to the field.

The technical contribution of this thesis is the implementation of both MST and Subsumptive Tabling optimization techniques in Micro-Datalog.

The empirical contribution is benchmarking across datasets and query patterns, comparing the implemented optimizations against pure semi-naive evaluation and Ascent, a state-of-the-art Datalog reasoner. The evaluation encompasses both synthetic datasets (dense and sparse graphs) and real-world data (Facebook social network), providing insights across different data characteristics and connectivity patterns.

Following this introduction, Chapter 2 provides background on relational algebra, Datalog semantics, and evaluation strategies, establishing the theoretical foundation for understanding the optimization techniques. Chapter 3 details the experimental methodology, including the implementation of MST and Subsumptive Tabling in Micro Datalog and the design of the benchmark suite. Chapter 4 presents experimental results across multiple datasets and query types, analyzing performance patterns and identifying optimal conditions for each optimization strategy. Finally, Chapter 5 concludes with a synthesis of findings, practical recommendations for Datalog system implementers, and directions for future research in query optimization for dynamic knowledge bases.

2. Background

2.1 Notation

$$\tau ::= \text{Const} \mid \text{Var}$$

$$a ::= P(\,\tau\,)$$

$$g ::= P\ (\text{Const}) \qquad (1)$$

$$\pi ::= a \leftarrow \{\,a\,\}$$

$$\Pi ::= \{\pi\}$$

$$E ::= \{\text{g}\}$$

Grammar 1 discerns predicates P , terms τ , rules π, atoms a, ground atoms g, programs Π, and finite fact stores E. { } represent finite, ordered sets of symbols. Constants are quoted names, and variables are unquoted. The left-hand side of a rule is called head, and the right-hand body.

Example 1. *Consider the Datalog rule*

$$\pi(\tau_1, \tau_2) \ = \ \text{tc}(\text{y}, 0) \ \leftarrow \ [\text{tc}(2, \text{y}), \text{tc}(\text{y}, 5)]$$

where

- tc *is a relation*
- τ_1, τ_2 *are terms in* tc

Let $a_1 = tc(2, y)$ and $a_2 = tc(y, 5)$ denote the first and second body atoms respectively, such that $body(\pi) = \{a_1, a_2\}$. The constant terms in a_1 are {2} at position {0}, and the constant terms in a_2 are {5} at position {1}.

2.2 Relational Algebra

Relational Algebra is a formal language used to query and manipulate relational databases, consisting of a set of operations over sets of tuples with fixed arity, i.e. relations. It provides a mathematical framework for querying databases [1].

Definition 1. A relation is a set of tuples with fixed arity, as shown in Example 1.

Definition 2. The selection operator $\sigma_{x=y}(R)$ returns a set of tuples from a relation *R*, such that every tuple has *y* as the typed value in the column at the position *x*.

Example 2. *For atom a_1, the selection operation $\sigma_{0=2}(tc)$ filters relation tc to retain only tuples where the first attribute equals to 2.*

$$\sigma_{0=2}(\mathrm{T}) \equiv \{t \in \mathrm{T} \mid t[0] = 2\}$$

For atom a_2, the selection operation $\sigma_{1=5}(T)$ filters relation T to retain only tuples where the second attribute equals to 5.

$$\sigma_{1=5}(\mathrm{T}) \equiv \{t \in \mathrm{T} \mid t[1] = 5\}$$

Definition 3. The join operation

$$R_x \bowtie_{z=w} R_y \equiv \{t_x \circ t_y \mid t_x \in R_x \wedge t_y \in R_y \wedge t_x[z] = t_y[w]\}$$

where:

- R_x and R_y are relations
- $t_x[z]$ denotes the value at position z of tuple t_x from R_x
- $t_y[w]$ denotes the value at position w of tuple t_y from R_y
- $t_x \circ t_y$ is a concatenation of the two tuples t_x and t_y

combines the relations R_x and R_y based on the shared variable at position z in R_x and w in R_y.

Example 3. *The join operation*

$$R_1 \bowtie_{1=0} R_2 \equiv \{t_1 \circ t_2 \mid t_1 \in R_1 \wedge t_2 \in R_2 \wedge t_1[1] = t_2[0]\}$$

where:

- $R_1 = \sigma_{0=2}(T)$ *with schema (2, y)*
- $R_2 = \sigma_{1=5}(T)$ *with schema (y, 5)*
- $t_1[1]$ *denotes the value at position 1 of tuple* t_1 *from* R_1
- $t_2[0]$ *denotes the value at position 0 of tuple* t_2 *from* R_2

combines the filtered relations based on the shared variable y, producing an intermediate result with schema (0, y, y, 5). After the join operation, the intermediate result contains attributes at positions [0, 1, 2, 3] corresponding to [0, y, y, 5].

Definition 4. The projection operation $\pi[t_0, t_1, ..., t_x](R)$ returns a relation R_π where:

- R is a relation that contains the intermediate result of an operation
- $t_0, t_1, ..., t_x$ denote the columns in R_π, such that t is either a constant or a value at a fixed position from relation R.

Example 4. *The projection operation*

$$\pi[\text{pos}(1), \text{const}(0)](\sigma_{0=2}(T) \bowtie_{1=0} \sigma_{1=5}(T))$$

maps the join result to the target schema T(y, 0).

2.3 The SQL Challenge with Graphs

Traditional relational databases organize data into tables, with SQL serving as the standard query language. However, many real-world problems involve complex relationships better modeled as graphs. Traditional SQL was not designed with recursion in mind. The relational algebra that forms the foundation of SQL lacks direct support for recursion, making it difficult to express queries that need to traverse hierarchical or graph-like data structures repeatedly to an unknown depth [1].

Example 5. *Consider a common structure for a simple social network where people can be friends:*

```
CREATE TABLE friends (
   person1_id INT,
   person2_id INT,
   PRIMARY KEY (person1_id, person2_id)
);

-- Finding friends of friends (2 hops away)
SELECT DISTINCT f2.person2_id AS friend_of_friend_id
FROM friends f1
JOIN friends f2 ON f1.person2_id = f2.person1_id
WHERE f1.person1_id = 1 AND f2.person2_id != 1;
```

The query becomes increasingly complex when trying to find relationships that are 3, 4, or n hops away.

The tabular paradigm of SQL makes it difficult to express the inherently recursive nature of path-finding in graphs. Although modern SQL implementations, such as SQL3 [8], have added extensions to handle recursion, they still have many limitations, such as only providing support for linear recursion, i.e. the FROM clause of the SELECT statement may include only one recursive relation [2].

2.4 Datalog's Graph-Native Approach

Datalog is a declarative query language that uses Horn clauses. It is based on first order logic and supports recursion. Its common use cases include describing systems and the construction of domain models.

Definition 5. EDB is extensional database and refers to relations not inferred by any rule, i.e. the ground truth[1]. IDB is intensional database and refers to relations defined by at least one rule.

Definition 6. An inferred fact is a tuple deduced from a rule.

[1] Ground truth refers to verified, true data [18].

Definition 7. A Datalog rule without a body is a ground fact, i.e. a tuple stored in EDB, and must only have constants as terms. It can also be referred to as a ground atom or a base atom.

Definition 8. A query is an atom $q(\{t\})$ where t can be a constant or a variable.

Definition 9. A Datalog program is a finite set of rules.

Example 6. *Consider the following program of a social network where people can be friends and a friend of a friend is inferred from the predicate reachable:*

```
person("Alice")
person("Bob")
person("Charlie")                                    (1)
friend("Alice", "Bob")
friend("Bob", "Charlie")

friends(x, y) ← friend(x, y)                         (2)
friends(x, y) ← friend(y, x)

reachable(x, y) ← friend(x, y)                       (3)
reachable(x, z) ← friend(x, y), friends(y, z)

reachable("Alice", _)                                (4)
```

(1) Ground facts. (2) Symmetric rules. (3) Transitive closure - finding all reachable people. (4) Query: Who can Alice reach?

Example 6 shows how Datalog offers a more natural way to represent and query graph structures.

To define the minimal model, we will use the wording of Paris Koutris [9]:

"Let P be a Datalog program. A pair of instances (I, J), where I is an EDB, and J is an IDB, is a model of P if (I, J) satisfies ΣP. Given an EDB I, the minimal model of P, denoted J = P(I), is a minimal IDB J such that (I, J) is a model of P."

"The semantics of a Datalog program P executed on EDB I is exactly the minimum model P(I)."

Each fact in Datalog can represent an edge or edges in a graph, while rules define how to traverse and derive new relationships from existing ones. The recursive rule for reachable

expresses path-finding logic that would require complex self-joins or recursive common table expressions in SQL.

2.5 Running example

A linear recursive rule is one where the recursive predicate appears at most once in the body of the rule [2]. A nonlinear recursive rule contains multiple calls to the same recursive predicate, which creates more complex evaluation patterns.

In Example 6, the predicate reachable was defined as a linear transitive closure, which iterates through all possible intermediate nodes to find paths between all pairs of nodes. A transitive closure program can also be nonlinear, which explores paths by recursively calling itself to find paths from a starting node to other nodes.

We define two predicates of transitive closure, linear and nonlinear, as a running example. The linear transitive closure predicate is defined in Datalog using the following two rules:

$$\mathrm{p}_1^{\mathrm{lin}} = \mathrm{tc}_{\mathrm{lin}}(\mathrm{x}, \mathrm{y}) \leftarrow \mathrm{e}(\mathrm{x}, \mathrm{y})$$
$$\mathrm{p}_2^{\mathrm{lin}} = \mathrm{tc}_{\mathrm{lin}}(\mathrm{x}, \mathrm{z}) \leftarrow \mathrm{e}(\mathrm{x}, \mathrm{y}), \mathrm{tc}_{\mathrm{lin}}(\mathrm{y}, \mathrm{z})$$
$$\Pi_{\mathrm{lin}} = \{\mathrm{p}_1^{\mathrm{lin}}, \mathrm{p}_2^{\mathrm{lin}}\}$$

The nonlinear transitive closure predicate is defined in Datalog with the following two rules:

$$\mathrm{p}_1^{\mathrm{rec}} = \mathrm{tc}_{\mathrm{rec}}(\mathrm{x}, \mathrm{y}) \leftarrow \mathrm{e}(\mathrm{x}, \mathrm{y})$$
$$\mathrm{p}_2^{\mathrm{rec}} = \mathrm{tc}_{\mathrm{rec}}(\mathrm{x}, \mathrm{z}) \leftarrow \mathrm{tc}_{\mathrm{rec}}(\mathrm{x}, \mathrm{y}), \mathrm{tc}_{\mathrm{rec}}(\mathrm{y}, \mathrm{z})$$
$$\Pi_{\mathrm{rec}} = \{\mathrm{p}_1^{\mathrm{rec}}, \mathrm{p}_2^{\mathrm{rec}}\}$$

There is a direct path from x to y if there is edge $\mathrm{e}(\mathrm{x}, \mathrm{y})$ and there is a path from x to z if one can go from x to y and from y to z.

Throughout the thesis, Rule 1 is used to refer to p_1 and Rule 2 to p_2 .

2.6 Evaluation Strategies

An evaluation strategy defines the procedure for how a program is evaluated during runtime.

A fixpoint refers to a state of the database where further application of the inference rules produces no new facts [1]. This represents the complete set of all possible derivations from the given rules and base facts, forming the minimal model of the Datalog program.

Bottom-up evaluation in Datalog is an evaluation strategy wherein computation begins with the explicit database facts (EDB) and iteratively applies inference rules to derive new facts (IDB) until a fixpoint is reached.

In contrast, top-down evaluation starts with the query predicate and recursively processes subgoals derived from rule bodies to determine whether the initial query is satisfiable. During this process, variable bindings from the original query are propagated downward through unification to narrow the search space.

2.6.1 Semi-Naive Evaluation

The standard evaluation procedure for Datalog is semi-naive evaluation, which is a type of bottom-up evaluation. It reduces redundant computation by only using newly derived facts in each iteration when applying recursive rules.

Consider the facts in EDB:

$$\mathrm{e}(\text{“a”}, \text{“b”})$$
$$\mathrm{e}(\text{“b”}, \text{“c”})$$
$$\mathrm{e}(\text{“c”}, \text{“d”})$$

And the query to compute all nodes with a path between them:

$$\mathrm{tc}_{\mathrm{rec}}(\_, \_)$$

Let P be a set of all known facts about $\mathrm{tc}_{\mathrm{rec}}$, Δ be newly derived facts in the last iteration and Q be a set of facts derived in the current iteration. By applying Rule 1 the first set of facts is collected and equal to the facts stored in the relation e in EDB.

$$\Delta_0 = \{(\mathrm{a}, \mathrm{b}), (\mathrm{b}, \mathrm{c}), (\mathrm{c}, \mathrm{d})\}$$
$$\mathrm{P} = \Delta_0$$

In the following iterations only Rule 2 is applied, since Rule 1 does not contain recursive predicates and therefore cannot deduce new facts. The condition for each iteration is that at least one body predicate in one rule is applied to at least one newly derived fact.

In the first iteration, Rule 2 is applied. In order to avoid recomputing derivations, only the set of already known facts P and newly derived facts Δ_0 are considered. Hence:

$$\mathrm{tc}_{\mathrm{rec}}(\mathrm{x}, \mathrm{y}) \in \Delta_0 \wedge \mathrm{tc}_{\mathrm{rec}}(\mathrm{y}, \mathrm{z}) \in \mathrm{P} \rightarrow \Delta_1 = \{(\mathrm{a}, \mathrm{c}), (\mathrm{b}, \mathrm{d})\}$$
$$\mathrm{P} = \mathrm{P} \cup \Delta_1$$

In the second iteration, Rule 2 is applied, such that

$$\text{tc\_rec}(x, y) \in \Delta_1 \land \text{tc\_rec}(y, z) \in P \rightarrow \Delta_2 = \{(a, d)\}$$
$$P = P \cup \Delta_2$$

In the third iteration, Rule 2 is applied, such that

$$\text{tc}_{\text{rec}}(x, y) \in \Delta_2 \land \text{tc}_{\text{rec}}(y, z) \in P \rightarrow$$
$$\text{tc}_{\text{rec}}(x, y) = \text{tc}_{\text{rec}}(\text{“a”}, \text{“d”}) \rightarrow$$
$$\text{tc}_{\text{rec}}(\text{“d”}, \_) \notin P \rightarrow$$
$$\Delta_3 = \emptyset$$

Δ_3 is empty, i.e. no new facts were inferred, hence a fixpoint is reached. The final result is

$$P = \{(a, b), (b, c), (c, d), (a, c), (b, d), (a, d)\}$$

The semi-naive algorithm guarantees completeness, i.e. all derivable facts will eventually be found, and termination.

Limitations of semi-naive evaluation

The most significant limitation of pure bottom-up evaluation is that it computes all derivable facts, regardless of whether they are relevant to a particular query. For instance, when computing transitive closure of a graph, a bottom-up evaluation will determine all reachable pairs, even if the user is only interested in whether one specific node can reach another.

For datasets with high connectivity or deep recursion, semi-naive evaluation can require substantial computational resources. Since it must compute the entire minimal model before any query can be answered, the initial computation overhead can be significant. For very large graphs the initial computation time can be longer than it is likely for the human race to survive.

2.6.2 SLD-Resolution

Selective Linear Definite clause resolution aka SLD-Resolution is a top-down evaluation strategy. It is a refutation-based method used to determine the satisfiability of a set of clauses. Given a query and a program, it uses subgoals to perform a goal-directed search through the program rules.

Example 7. *Consider a nonlinear transitive closure program and a query with only bound variables:*

$$tc_{rec}(\text{“a”}, \text{“d”})$$

Rule 1 does not infer any tuples, since there is no $e(\text{“a”}, \text{“d”})$ *in EDB.*

In Rule 2, the head of the rule $tc_{rec}(x, z)$ *is unified with the query, such that*

$$x = \text{“a”} \wedge z = \text{“d”}\ .$$

This yields the subgoals:

$$\{tc_{rec}(\text{“a”}, y), tc_{rec}(y, \text{“d”})\}$$

Let $\text{subgoal}_1 = tc_{rec}(\text{“a”}, y)$ *and* $\text{subgoal}_2 = tc_{rec}(y, \text{“d”})$.

Rule 1 and Rule 2 are applied to the subgoals.

$$tc_{rec}(\text{“a”}, y) \leftarrow e(\text{“a”}, y) \Rightarrow y = \text{“b”} \quad (1)$$

$$\text{subgoal}_3 = tc_{rec}(\text{“b”}, \text{“d”}) \quad (2)$$

$$tc_{rec}(\text{“b”}, \text{“d”}) \leftarrow tc_{rec}(\text{“b”}, y), tc_{rec}(y, \text{“d”}) \quad (3)$$

$$\text{subgoal}_4 = tc_{rec}(\text{“b”}, y), \text{subgoal}_5 = tc_{rec}(y, \text{“d”})$$

$$tc_{rec}(\text{“b”}, y) \leftarrow e(\text{“b”}, \text{“c”}) \Rightarrow y = \text{“c”} \quad (4)$$

$$\text{subgoal}_6 = tc_{rec}(\text{“c”}, \text{“d”}) \quad (5)$$

$$tc_{rec}(\text{“c”}, \text{“d”}) \leftarrow e(\text{“c”}, \text{“d”}) \quad (6)$$

(1) Applying Rule 1 to $subgoal_1$*. (2) Unifying subgoal_2 with y from* $subgoal_1$*. (3) Applying Rule 2 to* $subgoal_3$*. (4) Applying Rule 1 to* $subgoal_4$*. (5) Unifying* $subgoal_4$ *with y from* $subgoal_5$*. (6) Applying Rule 1 to* $subgoal_6$*. All subgoals have been satisfied.*

2.6.3 Optimization Strategies

Optimization strategies are used to improve the performance in a reasoner. An optimization strategy either transforms the program during its compilation to reduce the computation of redundant facts during runtime, or provides a strategy for materializing the program during runtime with the same goal.

In this section, a bottom-up evaluation compilation time strategy, Magic Sets Transformation, and a top-down evaluation runtime strategy, Subsumptive Tabling, are discussed.

Magic Sets Transformation

Magic sets transformation (MST) is a program transformation technique applied in the compilation time of the program that narrows down the search space based on the query and then evaluates the transformed program using a bottom-up evaluation strategy. It tries to mimic top-down evaluation without executing the program by inlining the query in the program as much as possible to reduce the amount of redundant facts computed.

Example 8. *Consider the linear transitive closure program with the following query:*

$$tc_{lin}(\text{“a”}, \_)$$

With the first argument a constant, i.e. bound, and the second argument free, the binding pattern can be added to the predicate, such that b denotes a bound argument and f denotes a free argument. This yields the predicate:

$$tc_{lin}\_bf$$

There are two steps in passing down information in the evaluation of a query. The first step is, given a constant in the query, unification of the query with the head of a rule, which binds some of the variables in the head atom to that constant. These bindings are passed down to the body of the rule as well. This is very similar to how top-down evaluation is done. [5]

Example 9. *Given the linear transitive closure program and the following facts:*

e("alice", "bob")
e("alice", "charlie")
e("bob", "david")
e("charlie", "eve")

For Rule 2:

1. *Unification: Query* tc_{lin}("alice", _) *unifies with head* tc_{lin}(x, z)
2. *Binding creation*: x = "alice", z *remains unbound*
3. *Binding propagation: The bindings are passed to the rule body:*
    a. e(x, y) *becomes* e("alice", y)
    b. tc_{lin}(y, z) *has* x *bound but* y, z *still unbound*

The second step is *sideways information passing* (SIP). Solving a predicate with the given bindings yields new bindings for the unbound variables, which can be passed to another predicate in the same rule to restrict the search space for the computation of that predicate [5].

A SIP-strategy defines the evaluation sequence for the predicates in the body of the rule [10]. The goal of a SIP-strategy are to order the body predicates in a way that minimizes computation cost. In order to achieve this, a SIP-strategy aims to fail as early as possible, call "cheaper" predicates before the expensive, and to minimize the size of the intermediate results of joins. Expensive predicates are, for example, recursive predicates, and predicates with complicated computations or other confounding factors [10]. SIP-strategy can be inlined by ordering the body literals of a rule already in the transformed program during compilation.

Example 10. *(continued) Consider Rule 1 of the linear transitive closure program, the query* tc_{lin}("alice", _) *and a SIP-strategy that maintains the original sequence of the body literals.*

1. *Solving the first predicate* e("alice", y) *yields the results:* y = "bob", y = "charlie".
2. *New bindings are generated: Two solutions with specific* y *values.*
3. *The* y *bindings are passed to the next predicate:*
    a. *First iteration:* tc_{lin}("bob", z) - *only explores paths from "bob"*
    b. *Second iteration:* tc_{lin}("charlie", z) - *only explores paths from "charlie"*

Without SIP tc_{lin}(y, z) *would explore all possible* (y, z) *pairs.*

Definition 10. An adorned predicate is a predicate with a binding pattern as a suffix. The binding pattern is considered the adornment.

Lemma 1. For every adorned predicate, a magic predicate is created with the purpose of restricting the search space to only the values that are relevant for the query. These magic predicates are added to the beginning of a body.

Example 11. *The MST transformed program of linear transitive closure.*

$$magic\_tc_{lin}\_bf(\text{“a”}) \quad (1)$$

$$magic\_tc_{lin}\_bf(y) \leftarrow magic\_tc\_lin\_bf(x), e(x, y) \quad (2)$$

$$tc_{lin}\_bf(x, y) \leftarrow magic\_tc_{lin}\_bf(x), e(x, y) \quad (3)$$

$$tc_{lin}\_bf(x, z) \leftarrow magic\_tc_{lin}\_bf(x), e(x, y), tc_{lin}\_bf(y, z) \quad (4)$$

(1) Magic seed ground fact. (2) Magic rule. (3) Transformed Rule 1 with a magic predicate. (4) Transformed Rule 2 with a magic predicate.

Example 11 shows the transformed rules of the linear transitive closure. With the original program, y would be evaluated for every value of x, and then tc_{lin} would be computed for every value of y. The transformed rule only considers the facts of e(x, y) where *x* is equal to *"a"*.

Example 12.

Given:

$x = \text{“b”} \rightarrow y = \text{“c”} \rightarrow tc_{lin}\_bf(\text{“c”}, \_)$ (1)

Then:

$magic\_tc_{lin}\_bf(\text{“c”}) \rightarrow$ (2)
$e(x, \text{“c”}) \rightarrow$
$x = \text{“b”} \rightarrow$
$magic\_tc_{lin}\_bf(\text{“b”})$ (3)

(1) From the evaluation of Transformed Rule 2. (2) From the evaluation of Magic Rule. (3) Magic predicate is saved in IDB.

Example 12 shows how magic predicates are updated to include only nodes reachable from *“a”,* which then restricts which edges are considered in the next iteration.

The magic predicates essentially track which facts are relevant based on the query pattern, simulating the way a top-down evaluation would only explore relevant paths but doing so before the runtime.

Theoretical Expectations for Magic Sets Transformation

Strengths. Magic sets have an advantage when the query bindings are highly selective, i.e. when the values provided in the query significantly narrow down the search space.

Let’s take the query *tc(“a”, _)* as an example. If *“a”* is one node out of millions in the database, then this binding is highly selective, since the search is restricted to only the paths from this one node, which is a very small portion of all the nodes in the database.

Another possibility could be the query *tc(_, “a”).* If *“a”* has very few paths leading to it compared to other nodes in the database, then this query would also be considered highly selective.

The selectivity of predicates within rule bodies as well as their placement in the bodies also impact performance. Magic sets have an advantage when predicates placed early in the rule bodies have high selectivity and the binding pattern allows these selective predicates to be evaluated with bound arguments.

Example 13. *Consider a query* $\mathrm{tc}_{\mathrm{lin}}(\text{“a”}, \_)$ *and the linear transitive closure program with the order of predicates in Rule 2 swapped, such that*

$$\mathrm{p}_2^{\mathrm{lin}} = \mathrm{tc}_{\mathrm{lin}}(\mathrm{x}, \mathrm{z}) \leftarrow \mathrm{tc}_{\mathrm{lin}}(\mathrm{y}, \mathrm{z}), \mathrm{e}(\mathrm{x}, \mathrm{y})$$

$\mathrm{tc}_{\mathrm{lin}}(\mathrm{y}, \mathrm{z})$ *is evaluated first with the binding pattern (free, free), which, depending on the size of the database, could generate thousands of intermediate results.* $\mathrm{e}(\mathrm{x}, \mathrm{y})$ *is evaluated second and must be joined against all the intermediate results.*

However, for a query with the second argument bound, i.e. $\mathrm{tc}_{\mathrm{lin}}(\_, \text{“a”})$, *this swapped version of the rule could be preferable.*

The earlier the candidate set for later predicates is reduced, the more efficient the evaluation process as this creates a compound filtering effect that progressively narrows the computation scope.

Magic sets also have an advantage when the query focuses on a localized region of the graph. Under these conditions, pure bottom-up would waste computation on irrelevant portions of the graph. This is especially important when memory constraints make exhaustive bottom-up evaluation impossible.

Limitations. For queries with low selectivity (e.g. "Find all pairs of nodes that can reach each other"), the overhead of MST might outweigh benefits since most facts end up being relevant anyway.

In densely connected graphs where most nodes can reach most other nodes, the propagation of magic predicates might approach the cost of full evaluation while adding overhead.

The transformation itself introduces additional predicates and rules. For simple programs or small datasets, this overhead might exceed the savings from reduced computation.

2.6.4 Subsumptive Tabling

Subsumptive tabling is a runtime optimization technique for top-down evaluation. It executes the program and outlines the mechanism for preventing the full recomputation of already derived facts.

During evaluation, answers to subqueries are stored in a table with their respective binding patterns that allows the reuse of the results instead of computation from scratch. This is the general technique of tabling.

While effective for identical subgoals, basic tabling still performs redundant computation for similar but non-identical subgoals. For example, after computing *tc("a", _),* computing *tc("a", "d")* would still be recomputed from scratch, despite being a more specific variant of a previously solved problem.

Subsumptive tabling introduces the concept of subsumption checking between subgoals. If results to a subgoal with a more general binding pattern have already been stored in the table, those results are filtered to obtain only those that match the more specific constraints of the current subgoal. If no subsuming subgoal exists, the subgoal is evaluated normally and its results and binding pattern are stored in the table.

The effectiveness of Subsumptive tabling relies a lot on the scheduling strategy, i.e. the order in which predicates are evaluated, since it may determine, whether a subquery is fully processed or can instead be evaluated using cached results in the table. This is because in one order of predicates a subsuming subquery may have been processed and tabled before and in another it may not have been [4].

Example 14. *Consider the query* tc(_, _) *and following rule with a constant:*

$$\mathrm{tc}(x, y) \leftarrow \mathrm{tc}(x, 1), \mathrm{tc}(y, x)$$

Using a depth-first, left-to-right scheduling strategy would evaluate all the possible values of tc(x, 1) *first and then proceed to* tc(y, x)*, which is a more general subquery and therefore could not make use of the subsumptive table. However, using right-to-left scheduling would evaluate* tc(y, x) *first, allowing the evaluation of* tc(x, 1) *to use the cached results from the table.*

Another variation in the algorithm pertains to whether early completion is used or not, and how a subquery is deemed suitable for completion. *No early completion* dictates the use of all relevant rules to infer answers. In contrast, *early completion* can stop evaluation for a subquery when the arguments are all bound and the subquery is evaluated to be true [11]. This, however, risks missing tuples in the final answer and requires an additional mechanism to handle it.

Theoretical Expectations for Subsumptive Tabling

Strengths. Like magic sets formations, the goal of subsumptive tabling is to eliminate redundant computation. Its ability to recognize when a new query can be answered using results from a previously computed, more general query, subsumptive tabling is efficient for workloads with hierarchical or overlapping queries. That is, unless the queries are evaluated in

the order of most specific to more general, which would render subsumptive tabling worse than useless (given the useless memory overhead).

Compared to variant tabling (which only reuses results for identical subgoals), subsumptive tabling can represent a broader range of query patterns with fewer table entries. By storing more general patterns and their results, the system can serve many specific variants without maintaining separate entries for each variant.

Limitations. The process of checking whether a new query is subsumed by existing tabled results introduces overhead. For simple queries with minimal recursion, this overhead might outweigh the benefits, especially if most queries are dissimilar enough that subsumption rarely occurs.

Workloads with highly diverse, unrelated queries benefit less from subsumptive tabling. If each query explores a completely different part of the data with little overlap, the subsumption mechanisms add overhead without providing optimization.

3. Experiment

The primary contribution of this thesis is empirical evaluation of the performance of MST and Subsumptive Tabling in incremental evaluation, compared against pure semi-naive evaluation and Ascent.

In the context of data processing, it is important to acknowledge that most real-world data is dynamic and a good reasoner should be able to make use of already inferred results when encountering new facts, instead of recomputing everything from scratch.

The technical contribution of this thesis is the implementation of MST and Subsumptive Tabling in Datalog, and the documentation of the process with the aim to bridge the gap in the practical research in Datalog.

This section is divided into two parts: the first outlines the implementation of MST and Subsumptive Tabling in Datalog, and the second part describes the benchmarks.

3.1 Datalog Implementation in Rust

Micro-Datalog [6] is an open-source minimal incremental semi-positive datalog reasoner implemented in Rust. It compiles Datalog rules into a sequence of relational algebra operations that are run incrementally with a four-instruction select-project-join relational algebra interpreter.

For the experiment of this thesis, magic set transformation and subsumptive tabling algorithms were implemented in the library along with unit tests and additional tools for running benchmarks.

3.1.1 Implementing Magic Set Transformation

Magic Sets Transformation is applied to a Datalog program based on a query and a program.

Definition 11. A magic set transformation is obtained from the function apply_magic_transformation: Π X q $\rightarrow \Pi_{magic}$, where the inputs are a program Π and a query q, and the output is a program Π_{magic}.

Definition 12. The function pred: a $\rightarrow$ str, when given some atom a as input, returns the predicate of the atom.

Example 15.

$\text{pred}(\text{tc}(x, y)) = \text{“tc”}$

Definition 13. The functions body: $\pi \rightarrow \{ a \}$ and head: $\pi \rightarrow a$, when given some program π as an input, return the respective body atoms, and head atom.

Example 16.

$\text{body}(p_1^{\text{lin}}) = \{e(x, y)\}$
$\text{head}(p_1^{\text{lin}}) = \text{tc}(x, y)$

Definition 14. An adorned atom $\text{adorn}(a, \beta)$, stems from some atom a, such that $\text{adorn}(a, \beta)$ has the same symbol as a, but with a suffix where each character corresponds to a character of the same index in β, with β as a list of characters, either *b* or f, for each term in a, denoting whether the term is bound or free.

Example 17.

$\text{adorn}(\text{tc}(\text{“a”}, \_), [\text{“b”}, \text{“f”}]) = \text{tc\_bf}(\text{“a”}, \_)$

Definition 15. A set of rules Π_{pred} of some predicate pred and some program Π contains all the rules $\pi \in \Pi \mid \text{pred}(\text{head}(\pi)) = \text{pred}$.

Example 18. *Since both* p_1^{lin} *and* p_2^{lin} *have* tc *as the predicate of their head atom,* Π_{tc} *is equal to* Π_{lin}.

Lemma 2. Let X be a set of atoms, such that for each atom $a \in X$ the function $\text{apply\_magic\_transformation}(\Pi\ X\ q)$ is invoked with a program Π and a query q as the inputs. X is initiated with an adorned atom $\text{adorn}(a, \beta)$, such that

$$a(t_1, .., t_n) = \text{head}(\pi) \mid \pi \in \Pi_{\text{pred}}(a)$$

$$\beta = \{b_1, \ldots, b_n\} := \begin{cases} b_n = \text{"b"} \; if \; t_n is \; a \; constant \\ b_n = \text{"f"} \; otherwise \end{cases}$$

Example 19. *For* Π_{rec}, X *would be initiated as* $X = \{tc_{rec}\_bf(x, y)\}$.

Definition 16. The position $pos(\pi, \Pi)$ of some rule π in some program Π is the index of π in Π.

Definition 17. The position $pos(a, \pi)$ of some body atom a in some rule π is the index of a in $body(\pi)$.

Example 20.

$pos(p_2^{lin}, \Pi_{lin}) = 1$

$pos(tc(x, y), p_2^{lin}) = 0$

Definition 18. A derived predicate pred is a predicate that appears in the head atom of at least one rule in the program, i.e. $pred \notin E$.

Example 21. p_2^{lin} contains two atoms with a derived predicate: tc(x, y) and tc(y, z).

Implementing Sideways Information Passing

Definition 19. An initial set of bound variables $T(\Pi, q)$ of some program Π, and some query q, is a set of all bound variables from q.

Definition 20. A set of bound variables of some program $\Pi = \{p_1, \ldots, p_i, \ldots, p_n\}$ is expanded from an initial set of bound variables T one subset of bound variables from one atom a in each p_i at a time, such that each $a \in p_i$ are analyzed in order. $T(pos(p, \Pi), pos(a, p))$ is then a set of variables that have been bound before pos(a, p) in the analysis of Π.

Lemma 3. Given some derived atom a | $a \notin E$, the binding pattern of an adorned atom adorn(a, β), β is determined based on the set of bound variables $T(pos(p, \Pi_a), pos(a, p))$, where:

- Π_a is the set of rules that define a
- p $\in \Pi_a$ is the rule being analyzed

Lemma 4. adorn(a, β) is added to the set of atoms to be processed X, given that at least one of the variables in adorn(a, β) is bound.

Lemma 5. Given some atom a | a ∈ E, if a includes a variable that is bound according to T(pos(π, Π_a), pos(a, pi)), then all the other terms in a are added to T(pos(π, Π_a), pos(π, pi)).

Example 22. *Let X be the set of atoms to be processed, i be the index of iterations in X_i and T be a set of bound variables for Π_{rec}.*

$$T(0, 0) = T \cup \{y\} = \{x, y\}$$
$$X_0 = \{tc_{rec}\_bf("a", \_)\} \quad (1)$$

$$T(1, 0) = T(0, 0) \quad (2)$$
$$X_1 = \{tc_{rec}\_bf("a", \_), tc_{rec}\_bf(x, y)\}$$

$$T(1, 1) = T(1, 0)$$
$$X_2 = \{tc_{rec}\_bf("a", \_), tc_{rec}\_bf(x, y)\} \quad (3)$$

(1) No adorned atoms were added to X, since Rule 1 does not contain derived atoms.
(2) No bound variables were added to T, since $tc_{rec}(x, y)$ is a derived predicate.
(3) $tc_{rec}\_ff(y, z)$ has both variables free and is therefore not added to X.

Creating magic rules

Definition 21. A magic atom magic(a) of an atom a, such that adorn(a, β) is the adorned version of a, is an atom that has the same β as a and has the symbol of adorn(a, β) with the prefix *magic_* and only bound variables as terms.

Example 21.

Given:

$\text{adorn}(tc_{rec}(x, y), \text{'bf'})$

Then:

$\text{magic}(tc(x, y)) = \text{magic\_}tc_{rec}\text{\_bf}(x).$

Definition 22. A binding chain $B(\pi, a)$ of some rule π and some atom $a \in \text{body}(\pi)$ is a set of atoms, such that

$$B(\pi, a) = \{\text{magic}(\text{head}(\pi)), \text{adorn}(c_0, \beta_0), \ldots, \text{adorn}(c_{pos(a,\pi)} - 1, \beta_{pos(a,\pi)-1})\} \mid c \in \text{body}(\pi)$$

Example 22.

Given:

$a = tc_{rec}(y, z)$

Then:

$B(\pi_{rec}, a) = \{ \text{magic\_}tc_{rec}\text{\_bf}(x),\ tc_{rec}\text{\_bf}(x, y)\}$

Lemma 6. For each atom $a \in \text{body}(\pi) \mid p(a) \notin E \wedge \text{magic}(a) \notin B(\pi, a)$, given that a includes a bound term, a magic rule $\pi_{magic}(a)$ is created with $\text{magic}(a)$ as the head and $B(\pi, a)$ as the body.

Example 23.

Given:

$\alpha = tc_{rec}(y, z)$

Then:

$\pi_{magic}(\alpha) = \text{magic\_}tc_{rec}\text{\_bf}(y) \leftarrow \{\text{magic\_}tc_{rec}\text{\_bf}(x), tc_{rec}\text{\_bf}(x, y)\}$

Definition 23. A modified rule body is obtained from the function modify_rule_body: $\pi \rightarrow \text{body}(\pi)_{\text{mod}}$, where the inputs are the program Π, query q, the rule $\pi \in \Pi$, and the output is a set of adorned atoms $\text{adorn}(\alpha, \beta) \mid a \in \text{body}(\pi)$ with a magic rule $\pi_{\text{magic}(\alpha)}$ added in front of $\text{adorn}(\alpha, \beta)$, given that $\text{pred}(\alpha) \notin E$ and α contains at least one bound term.

Definition 24. A modified rule is obtained from the function modify_rule: $\pi \rightarrow \pi\_\text{mod}$, where the input is the rule π, and the output is the modified rule π_{mod}, such that

$$\text{head}(\pi_{\text{mod}}) = \text{adorn}(\text{head}(\pi), \beta) \Lambda\, \text{body}(\pi_{\text{mod}}) = \text{modify\_rule\_body}(\pi)$$

Example 24.

Given:

$\pi_1, \pi_2 \in \Pi_{\text{rec}}$

Then:

$\text{modify\_rule}(\pi_1) = \text{tc}_{\text{rec}}\text{\_bf}(x, y) \leftarrow \{\text{magic\_tc}_{\text{rec}}\text{\_bf}(x), e(x, y)\}$

$\text{modify\_rule}(\pi_2) = \text{tc}_{\text{rec}}\text{\_bf}(x, z) \leftarrow \{\text{magic\_tc}_{\text{rec}}\text{\_bf}(x), \text{tc}_{\text{rec}}\text{\_bf}(x, y), \text{magic\_tc}_{\text{rec}}\text{\_bf}(y), \text{tc}_{\text{rec}}\text{\_bf}(y, z)\}$

An optimization technique of transformed rules is introduced in [5]: a magic predicate in the body of π_{magic} can be deleted unless it corresponds to the bound arguments of the head of π.

Thus

$$\text{tc}_{\text{rec}}\text{\_bf}(x, z) \leftarrow \{\text{magic\_tc}_{\text{rec}}\text{\_bf}(x), \text{tc}_{\text{rec}}\text{\_bf}(x, y), \text{magic\_tc}_{\text{rec}}\text{\_bf}(y), \text{tc}_{\text{rec}}\text{\_bf}(y, z)\}$$

becomes

$$\text{tc}_{\text{rec}}\text{\_bf}(x, z) \leftarrow \{\text{magic\_tc}_{\text{rec}}\text{\_bf}(x), \text{tc}_{\text{rec}}\text{\_bf}(x, y), \text{tc}_{\text{rec}}\text{\_bf}(y, z)\}$$

When we consider that

$$\text{magic\_tc}_{\text{rec}}\text{\_bf}(y) \leftarrow [\text{magic\_tc}_{\text{rec}}\text{\_bf}\ (x), \text{tc}_{\text{rec}}\text{\_bf}(x, y)]$$

and replace the head with the body of the rule in the modified rule pi_mod, we get:

$$\text{tc}_{\text{rec}}\text{\_bf}(x, z) \leftarrow \{\text{magic\_tc}_{\text{rec}}\text{\_bf}(x), \text{tc}_{\text{rec}}\text{\_bf}(x, y), \text{magic\_tc}_{\text{rec}}\text{\_bf}(x), \text{tc}_{\text{rec}}\text{\_bf}(x, y), \text{tc}_{\text{rec}}\text{\_bf}(y, z)\}$$

This renders magic_tc_{rec}_bf(y) redundant in π_{mod}.

Definition 25. $\Pi_{magic(\pi, q)}$ of some rule $\pi \in \Pi$ and some query q is a set of magic rules derived from π. $\Pi_{magic(\Pi, q)}$ is a set of $\Pi_{magic(\pi, q)}$ for every $\pi \in \Pi_{pred(q)}$.

Example 25.

Given:

$\Pi = \Pi_{lin}$

$q = tc(\text{“a“}, \_)$

Then:

$\Pi_{magic(\Pi, q)} = \{magic\_tc_{lin}\_bf(y) \leftarrow [magic\_tc_{lin}\_bf(x), e(x, y)]\}$

Definition 26. $\Pi_{mod(\pi, q)}$ of some rule $\pi \in \Pi$ and some query q is a set of modified rules derived from π. $\Pi_{mod(\Pi, q)}$ is a set of $\Pi_{mod(\pi, q)}$ for every $\pi \in \Pi_{pred(q)}$.

Example 26.

Given:

$\Pi = \Pi_{lin}$

$q = tc(\text{“a“}, \_)$

Then:

$\Pi_{mod(\pi, q)} = \{$

$tc_{lin}\_bf(x, y) \leftarrow [magic\_tc_{lin}\_bf(x), e(x, y)],$

$tc_{lin}\_bf(x, y) \leftarrow [magic\_tc_{lin}\_bf(x), e(x, y), tc_{lin}\_bf(y, z)]$

$\}$

Definition 27. A complete magic transformation is obtained from the function apply_magic_transformation$(\Pi \, X \, q) \rightarrow \Pi_{magic}$, where the inputs are a program Π and a query q, and the output is a set of rules $\Pi_{magic} = \Pi_{magic(\Pi, q)} \cup \Pi_{mod(\Pi, q)}$.

3.1.2 Implementing Subsumptive Tabling

Subsumptive tabling is an optimization technique in top-down evaluation that uses a cache table to narrow the search for subqueries that are subsumed by the cached subqueries.

Lemma 7. A query is presented as an atom $q(\{\tau\})$ where q is the predicate and $\{\tau\}$ is a set of terms.

Definition 28. Let subsumptive table $T<q, s>$ be a set of key-value tuples where the key q is a query atom and the value s is a complete set of inferred tuples.

Definition 29. Atom $q_1(\{\tau^1\})$ subsumes atom $q_2(\{\tau^2\})$, where $\{\tau^1\}$ and $\{\tau^2\}$ are sets of terms of equal length, if one of the following conditions is met:

1) $\tau^1_i = \tau^2_i \mid \tau^1_i = \text{Const} \wedge \tau^2_i = \text{Const}$
2) $\tau^1_i = \text{Const} \wedge \tau^2_i \neq \text{Const}$
3) $\tau^1_i \neq \text{Const}$

for every $\tau^1_i \in \{\tau^1\} \wedge \tau^2_i \in \{\tau^2\}$.

Example 27.

Given:

$q_1 = tc(\_, 2)$

$q_2 = tc(3, 2)$

Then:

q_1 subsumes q_2

Lemma 8. Before evaluating the subquery, the subsumptive table is checked for subsuming query patterns and if one is found, then its cached results are used instead of processing the subquery.

Definition 30. A binding is a tuple (v, c) where v is the name of a variable and c is its constant value.

Definition 31. $B(\pi) := \{(v, c)\}$ is a set of bindings of the rule π, where v is a variable and c is the value that holds for v throughout the rule π.

Definition 32. $\Pi(\alpha)$ is a set of rules for atom q, such that for every $\pi \in \Pi$, head(π) subsumes α.

Example 28.

Given:

$\pi_0 = \text{tc}(x, y) \leftarrow \text{e}(x, y)$
$\pi_1 = \text{tc}(x, 1) \leftarrow \text{e}(x, 1)$
$\pi_2 = \text{tc}(2, y) \leftarrow \text{e}(2, 1)$
$\alpha = \text{tc}(\_, 2)$

Then:

$\Pi(\alpha) = \{\pi_0, \pi_2\}$

Lemma 9. The rules in $\Pi(\alpha)$ and the body atoms in $\pi \in \Pi$ are evaluated sequentially. After every complete evaluation of an atom, the set of inferred tuples is stored in the subsumptive table.

Definition 33. Given a query atom α^q with a set of terms $\{\tau^q\}$, for every rule $\pi \in \Pi(\alpha^q)$ with a set of terms $\{\tau^h\}$ in head(π), a subquery atom $\alpha^q{}_{sub}$ with a set of terms $\{\tau^{sub}\}$ is generated, such that

- $\tau^{sub}{}_i = \tau^q{}_i$, if $\tau^q{}_i$ is a constant,
- otherwise $\tau^{sub}{}_i = \tau^h{}_1$.

Example 29.

Given:

$\alpha^q = \text{tc}(1, z)$

$\pi = \text{tc}(x, y) \leftarrow \text{e}(x, y)$

Then:

$\alpha^q{}_{sub} = \text{tc}(1, y)$

Lemma 10. For every atom α in body(π_a) | $\pi_a \in \Pi(\alpha^q) \wedge a \notin E$, where α^q is a query atom, the bound variable terms in α are replaced with constants according to the set of bindings B(π_a) to create a subquery atom $\alpha^q{}_{sub}$.

Example 30.

Given:

$\pi_a = \text{tc}(x, z) \leftarrow \text{tc}(x, y), \text{tc}(y, z)$

$\Pi(\alpha^q) = \{\pi_a\}$

$\alpha = \text{tc}(x, y)$

$B(\pi_a) = \{x: 1\}$

Then:

$\alpha^q{}_{sub} = \text{tc}(1, y)$

Lemma 11. For every inferred tuple (a, b) from the evaluation of the subquery atom $\alpha^q{}_{sub}$, $B(\pi_a)$ is updated and used to evaluate the subsequent atoms in $\text{body}(\pi_a)$.

Example 31.

Given:

$(a, b) = (1, 3)$

Then:

$B(\pi_a) = \{x: 1, y: 3\}$

3.2 Benchmarks

Ascent [7] is a bottom-up Datalog reasoner implemented in Rust and is used for comparison in the benchmarks as the baseline reasoner.

Datasets. Four datasets were used for the benchmarks. The first dataset was LUBM [12], a synthetic dataset commonly used to evaluate the performance of logical reasoning and inference when working with knowledge represented in description logic-based languages like RDFS [13] and OWL2RL [14]. The dataset consists of two parts: an ontological framework that specifies the available predicates for describing individuals, and the descriptions of the individuals themselves.

$$T(y,\text{"rdf:type"}, x) \leftarrow T(a, \text{"rdfs:domain"}, x), T(y, a, z) \quad (1)$$
$$T(z, \text{"rdf:type"}, x) \leftarrow T(a, \text{"rdfs:range"}, x), T(y, a, z)$$
$$T(x, \text{"rdfs:subPropertyOf"}, z) \leftarrow T(x, \text{"rdfs:subPropertyOf"}, y), T(y, \text{"rdfs:subPropertyOf"}, z)$$
$$T(x, \text{"rdfs:subClassOf"}, z) \leftarrow T(x, \text{"rdfs:subClassOf"}, y), T(y, \text{"rdfs:subClassOf"}, z)$$
$$T(z, \text{"rdf:type"}, y) \leftarrow T(x, \text{"rdfs:subClassOf"}, y), T(z, \text{"rdf:type"}, x)$$
$$T(x, b, y) \leftarrow T(a, \text{"rdfs:subPropertyOf"}, b), T(x, a, y)$$

The RDFS entailment program is shown in equation 1.

$$\text{Employee(x)} \leftarrow \text{Faculty(x)} \quad (2)$$
...
$$\text{Professor(x)} \leftarrow \text{Dean(x)}$$
...
$$\text{subOrganizationOf(x, z)} \leftarrow \text{subOrganizationOf(x, y)}, \text{subOrganizationOf(y, z)}$$

Following the instructions of [15], the description logic entailments of LUBM's ontology are converted to Datalog, producing the OWL2RL entailment program, which consists of 96 rules, as shown in equation 2.

The second and third datasets were created using the GT [16] tool. RAND1K is a densely connected graph containing one thousand edges, where each edge maintains approximately a 1% probability of connection to other edges. RMAT1K is a sparse graph with ten thousand edges and one thousand nodes, characterized by an inverse power-law distribution where the majority of nodes are connected to only a small number of other nodes.

The fourth dataset is an anonymized real-life dataset of "circles" (or "friends' lists") from Facebook [17]. It is a very large sparse graph of almost 90,000 edges and four thousand nodes.

$$TC(x, y) \leftarrow E(x, y) \quad (3)$$

$$TC(x, y) \leftarrow TC(x, y), TC(y, z)$$

$$TC(x, y) \leftarrow E(x, y) \quad (4)$$

$$TC(x, y) \leftarrow E(x, y), T(y, z)$$

The non-linear transitive closure program in equation 3 is run on the second graph, RAND1K, and the linear transitive closure program in equation 4 is run on the third and fourth graph, RMAT1K and Facebook.

All benchmarks were run on the same M3 Macbook Air. No other processes were running during the evaluation. Each benchmark was run 25 times, unless stated otherwise. The time

taken for the materialisation of a program for was measured with the struct Instant from the Standard Rust Library[2].

Queries. Three queries were evaluated with every dataset and program combination. The first query was chosen with the aim to provide the best case scenario for MST and Subsumptive Tabling. These queries have all terms bound with the most common values in the dataset. The second query has one term bound and has a result of <1% of the data. The third query has no bound terms.

[2] The Rust Project Developers, "std::time::Instant," The Rust Standard Library, Aug. 6, 2025. [Online]. Available: https://doc.rust-lang.org/std/time/struct.Instant.html

4. Results

The first benchmark set is shown on Figure 1. The X-axis shows the number of edges given to the reasoner to materialize, according to the respective program and dataset, which are indicated by the title of each graph. The Y-axis shows the time in microseconds taken to materialize the program.

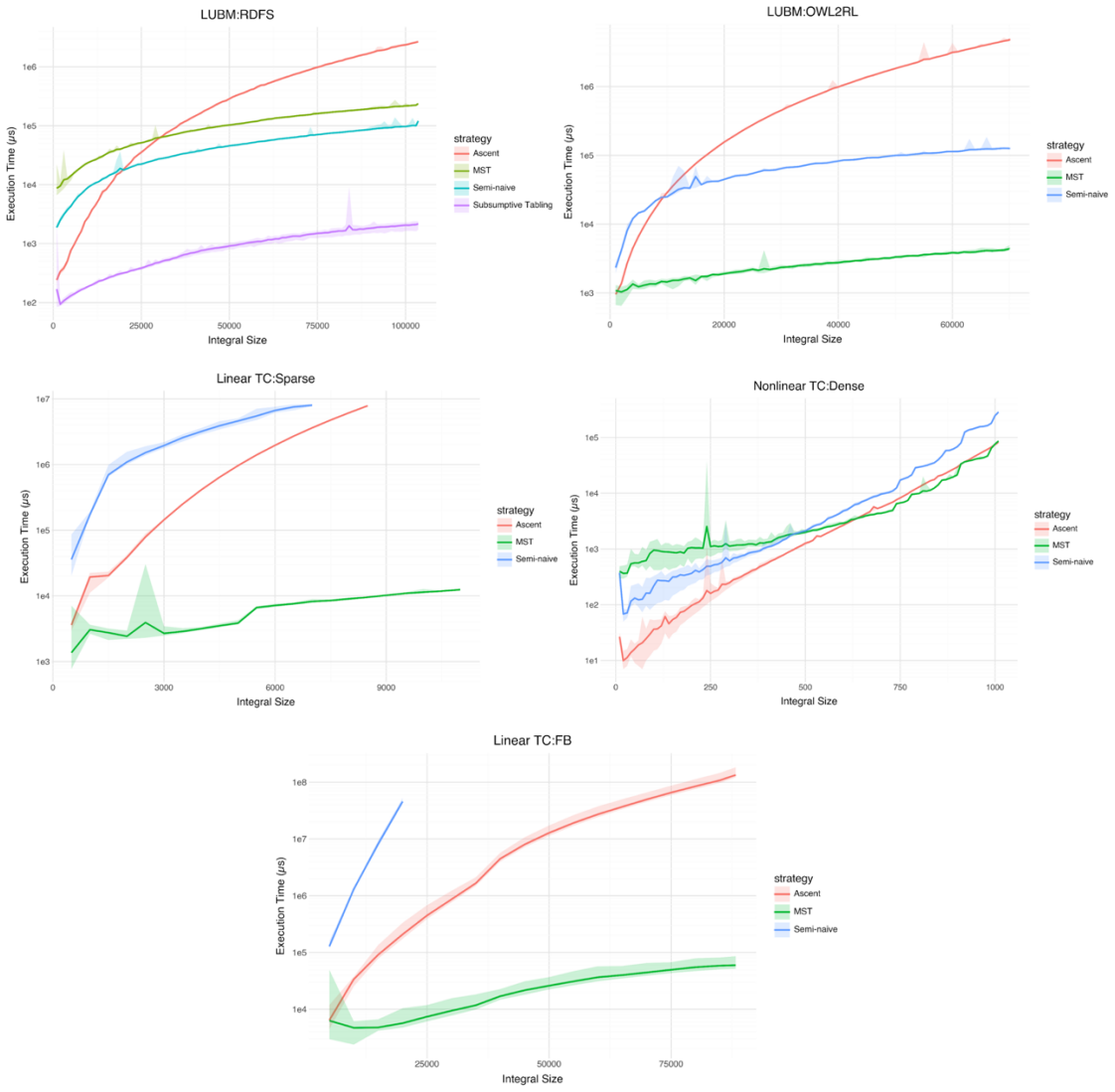

Figure 1. The performance of pure semi-naive evaluation, MST, Subsumptive Tabling and Ascent with a query with all terms bound.

With all terms bound in the query, in two out of the five graphs, MST is consistently faster than both pure semi-naive evaluation and Ascent. In two out of the remaining three graphs, it reaches or surpasses the performance of semi-naive and Ascent as the amount of data is increased. In

Linear TC:FB, MST becomes over 2000 times faster than Ascent in the last iteration. Subsumptive tabling is the fastest in LUBM:RDFS. In Nonlinear TC:Dense surpasses the performance of Ascent as the amount of data is increased and remains faster.

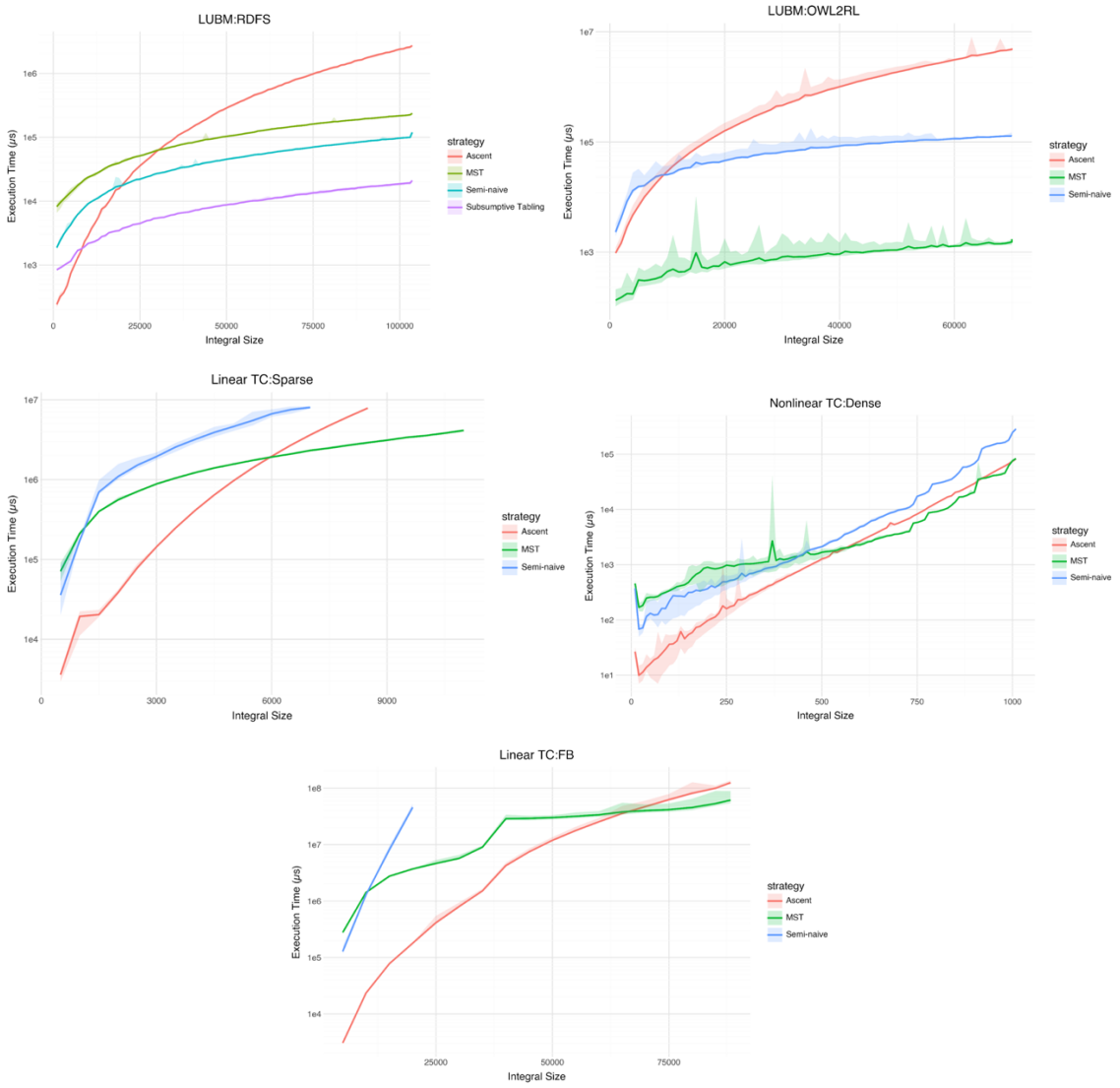

Figure 2. The performance of pure semi-naive evaluation, MST, Subsumptive Tabling and Ascent with a query with one term bound.

As shown in Figure 2, with only one term bound in the query, notable changes occur in Linear TC:Sparse and Linear TC:FB with a decrease in the performance of MST, which surpasses Ascent only once the data has increased at least twofold.

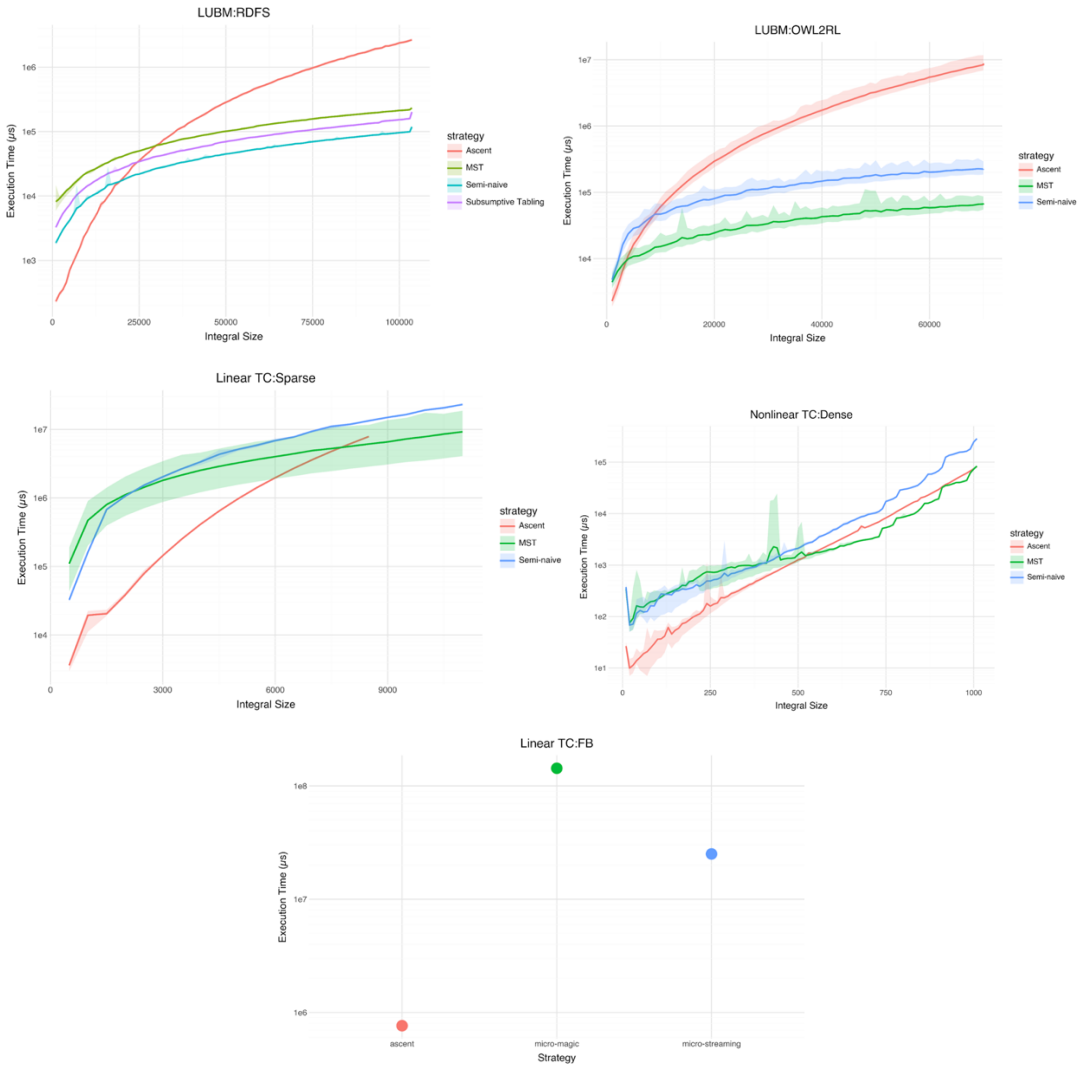

Figure 3. The performance of pure semi-naive evaluation, MST, Subsumptive Tabling and Ascent with a query with all terms free.

Figure 3 shows that with all terms free in the query, MST can be considered useless for a program on a graph as large as sparse Linear TC:FB. In Linear TC:Sparse its average performance briefly surpasses Ascent but the performance becomes a lot less uniform. While the performance of MST shows a large incline in LUBM:OWL2RL, it remains faster than Ascent and pure semi-naive.

The benchmarks with LUBM:RDFS show that Subsumptive Tabling has the greatest advantage when the query terms are bound, and with all query terms free, it loses its advantage over pure semi-naive evaluation. This demonstrates a similar tendency to MST in favoring highly selective queries. With all terms bound it achieves performance that is 1000 times faster than Ascent.

LUBM:OWL2RL and LUBM:RDFS do not require multiple iterations to converge and in both of the graphs, the differences in performance between the different queries are the most minimal for both MST and Subsumptive Tabling.

The advantage of MST is the most evident in sparse graphs with a linear transitive closure program, i.e. Linear TC:Sparse and Linear TC:Facebook. In Linear TC:Sparse the evaluation of the query with all terms bound is over 300 times faster than of the query with only one term bound, and over 700 times faster than of the query with no terms bound. In Linear TC:Facebook, the respective ratios are one thousand and two thousand.

The results vary greatly when the nonlinear transitive closure program is used to evaluate a sparse graph instead of the linear transitive closure. Evaluating only the first batch of 2000 edges of RMAT1K with the nonlinear program took MST over *50 seconds*, which is over five thousand times slower than the slowest evaluation with the linear program of the same graph.

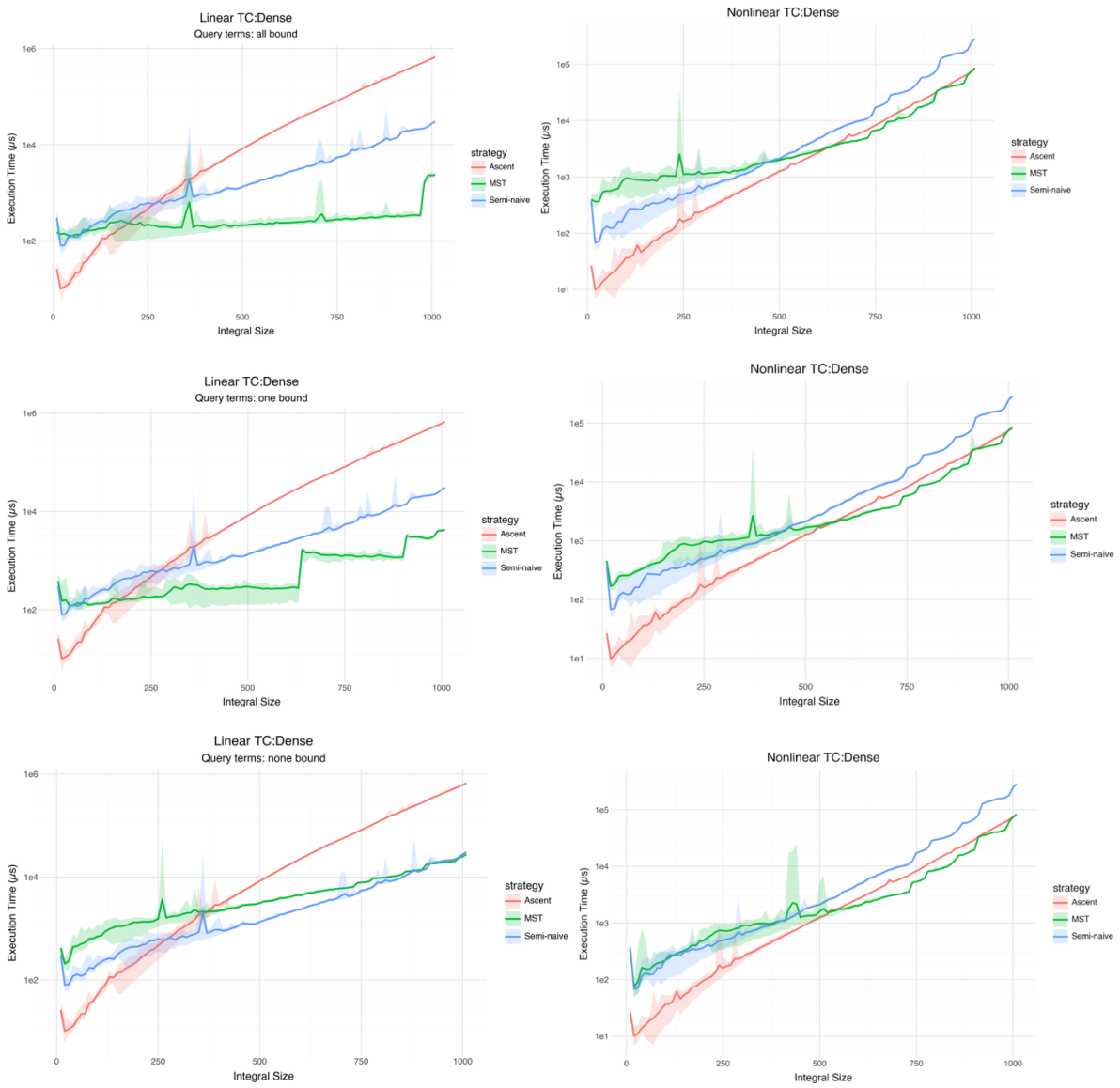

Figure 4. The performance of pure semi-naive evaluation, MST, and Ascent on RAND1K with a linear transitive closure program and queries with different binding patterns.

Compared to its nonlinear counterpart, the plots for the performance of MST with linear transitive closure vary a lot more (as seen in Figure 4). In all cases, MST surpasses Ascent earlier than in nonlinear but does not pass pure semi-naive evaluation with no bound query terms, which aligns with expectations.

An interesting leap in performance time occurs in Linear TC:Dense with one query term bound between batch sizes 630 and 640. The gathered data shows that the number of inferred tuples at batch size 630 is 0 and at batch size 640 is 14. This leap could be due to the increase in the stored magic facts in IDB.

One contributing factor to the difference in MST's performance in linear and nonlinear programs could lie in their program transformations (A full transformation of the programs is presented in Appendix A).

Table 1. Generated modified rule predicates across program types and query binding patterns.

Program and query binding pattern	**Linear:BB**	**Linear:BF**	**Nonlinear:BB**	**Nonlinear:BF**
Modified rule predicates	tc_bb tc_bf	tc_bf	tc_bb tc_bf tc_ff	tc_bf tc_ff

Table 1 shows that tc_ff modified rule predicate is created in the nonlinear program with both query binding patterns, unlike in the linear program, where it is absent. Since tc_ff has both terms free, it is essentially the same as an unmodified rule and offers no restrictions in search space.

The predicate tc_ff is called as the last body predicate in tc_bf. This means that the values passed to tc_ff have still gone through some filters by the bindings in tc_bf, therefore not every possible binding gets processed with tc_ff, which decreases the workload to an extent compared to pure semi-naïve evaluation. This is illustrated in the plots of Nonlinear TC:Dense where MST passes pure semi-naïve evaluation but does not offer a remarkable advantage.

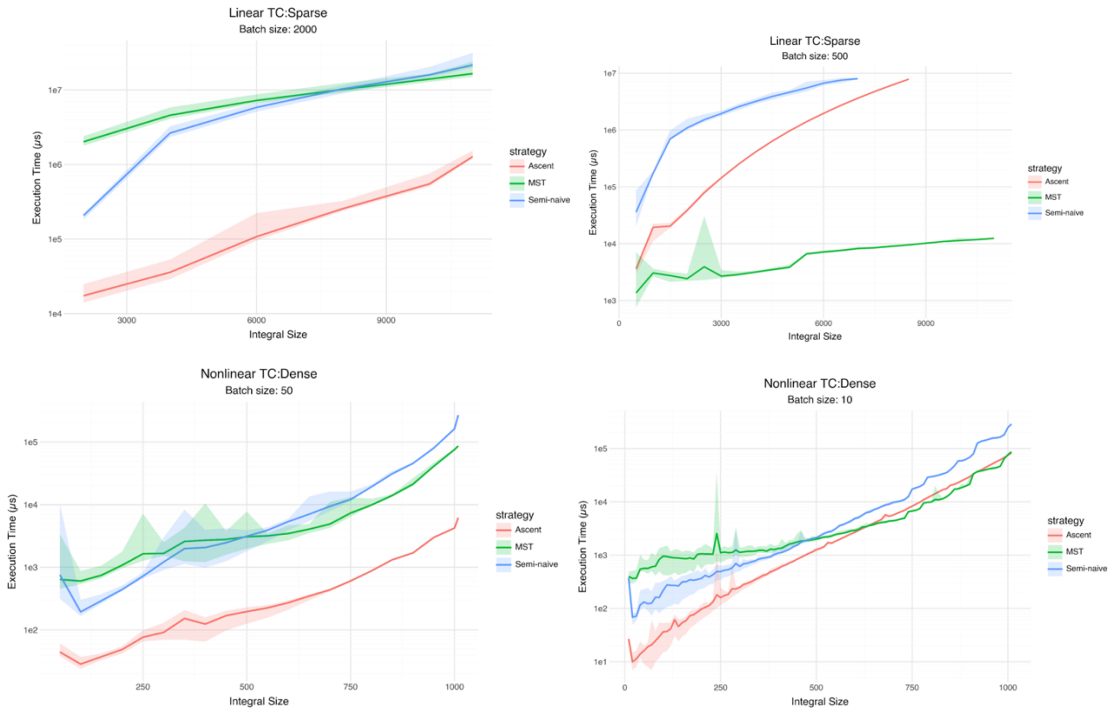

Figure 5. The performance of pure semi-naive evaluation, MST and Ascent with different batch sizes.

The factor with a strong impact on the advantage of MST over Ascent and pure semi-naive evaluation is the batch size, as shown in Figure 5 – the smaller the batches and larger the cumulation, the greater the advantage of MST. The three-way comparison of MST, pure semi-naive evaluation and Ascent show exactly the value MST adds, since without it, Ascent is faster than pure semi-naive evaluation even with smaller batches of data. This shows that the advantage of MST cannot be solely attributed to incremental evaluation, and its narrowing of the search space plays a decisive role in its performance.

4.1 Stress test for the compiler

MST can increase the size of the original program manyfold, depending on the number of rules in the original program, the number of body atoms in the rules, the arity of all the atoms in the program, and the query. For example, the transformed program of RDFS, which has 7 rules, with a query with all terms bound is ~100 rules.

The benchmarks on LUBM:RDFS and LUBM:OWL2RL also served as a fuzz test for the compiler due to the size of the transformations. The correctness of the compiler was tested by validating its output against that of Ascent.

A number of bugs were found. One of the most significant was the discovery that the compiler was vulnerable to the order of atoms – it expected two atoms next to each other in evaluation to have at least one variable in common. Since a proper fix would have been too lengthy for the scope of this thesis, a hack of canonical rule ordering was used.

5. Conclusion

The two primary contributions of this thesis are to the field of Datalog query optimization. The technical contribution involves implementing both MST and Subsumptive Tabling in Micro Datalog, a Rust-based incremental reasoner. This implementation fills a significant gap in available Datalog systems, as most existing research remains theoretical without accessible practical implementations. The documented implementation process serves as a resource for future Datalog system developers.

The empirical contribution provides a comparative evaluation of these optimization strategies in incremental evaluation scenarios. The experiment uses benchmarks across synthetic datasets (dense and sparse graphs), real-world data (Facebook social network), and knowledge base reasoning tasks (LUBM with RDFS and OWL2RL. The evaluation methodology, incorporating streaming data with varying batch sizes, more accurately reflects the dynamic nature of real-world applications than traditional static benchmarks.

The comparison between Magic Sets Transformation and pure semi-naïve evaluation allows us to separate the exact effect of the optimization strategy. The comparison with Subsumptive Tabling gives more context for the comparison by using top-down evaluation, and the comparison with Ascent gives credibility that the shown differences are not specific to one reasoner only.

The results show that the effectiveness of Datalog optimization strategies depends on the interplay between data characteristics, query patterns, and program structure. Magic Set Transformation performs the best for highly selective queries on sparse graph structures, particularly when using linear transitive closure programs, where it achieves speedups of several orders of magnitude compared to both pure semi-naive evaluation and competing reasoners.

Subsumptive Tabling demonstrates superior performance in LUBM:RDFS and is up to 1000 times faster than Ascent for a query with all terms bound. While its effectiveness declines with free query terms, its performance remains on bar with the other strategies.

Both optimization techniques exhibit significant performance degradation under certain conditions. MST shows poor performance with unbound queries on dense graphs and struggles particularly with nonlinear recursive programs on sparse data. The transformation overhead

can exceed the computational savings when query constraints provide insufficient selectivity to reduce the search space effectively.

The research also shows the impact of incremental evaluation patterns on optimization effectiveness. Smaller batch sizes consistently favor MST over traditional bottom-up approaches, suggesting that the technique's benefits compound in scenarios with frequent updates — a common characteristic of modern data-intensive applications.

Based on the empirical findings, this research provides concrete guidance for practitioners choosing between Datalog optimization strategies:

Magic Set Transformation is most effective for scenarios involving highly selective queries with multiple bound terms and specific values, particularly when working with sparse graph structures and linear recursive programs. MST shows advantages when processing incremental updates in small batches.

Subsumptive Tabling performs optimally with query workloads that exhibit hierarchical or overlapping patterns, especially when queries typically have bound terms but vary in their level of specificity. Complex programs with multiple interdependent rules also favor Subsumptive Tabling.

Both optimization strategies should be avoided in certain scenarios where their overhead exceeds their benefits. When queries are predominantly unbound and require full materialization of results, the optimization overhead typically outweighs any computational savings. Similarly, dense graphs where most facts are relevant to the query provide little opportunity for pruning, making the transformation costs unjustifiable. Simple programs with minimal recursion often perform better with pure semi-naive evaluation.

This research focused primarily on transitive closure and knowledge base reasoning tasks, which represent important but limited subsets of Datalog applications. Future research should extend the evaluation to broader classes of recursive programs, including more complex graph algorithms, program analysis tasks, and distributed reasoning scenarios.

In future work, Subsumptive Tabling could also be used in transitive closure benchmarks to add a comparison of top-down evaluation.

The batch size analysis revealed interesting patterns but was limited to relatively simple update patterns. Future work should investigate more complex incremental scenarios, including

deletions, batch reordering, and adaptive batch sizing strategies that could dynamically adjust based on query characteristics and data patterns.

Additionally, the thesis identified several implementation challenges in Micro-Datalog, particularly around rule ordering dependencies in MST. Currently the challenges were fixed with “hacks“ but a proper rework of the code could unlock further performance improvements.

References

[1] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* vol. 13, no. 6, pp. 377-387, 1970.

[2] T. Pieciukiewicz, K. Subieta and K. J. Stencel, "Recursive Query Processing in SBQL," in *Object Databases, First International Conference*, Berlin, 2008.

[3] S. Ceri, G. Gottlob and L. Tanca, "What You Always Wanted to Know About Datalog (And Never Dared to Ask)," *Transactions on Knowledge and data Engineering,* vol. 1, no. 1, 1989.

[4] K. T. Tekle and Y. A. Liu, "More Efficient Datalog Queries: Subsumptive Tabling Beats Magic Sets," in *SIGMOD '11: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011.

[5] C. Beeri and R. Ramakrishnan, "On the power of magic," in *The sixth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1987.

[6] B. R. Carneiro Alves de Lima, "Micro-Datalog," [Online]. Available: https://github.com/brurucy/micro-datalog. [Accessed 3 August 2025].

[7] A. Sahebolamri, T. Gilray and K. K. Micinski, "Seamless deductive inference via macros," in *31st ACM SIGPLAN International Conference on Compiler Construct*, 2022.

[8] L. Libkin, "Expressive power of SQL," *Theoretical Computer Science,* vol. 296, no. 3, pp. 379-404, 2003.

[9] P. Koutris, "Lecture 7: Introduction to Datalog," Spring 2016. [Online]. Available: https://pages.cs.wisc.edu/~paris/cs838-s16/lecture-notes/lecture7.pdf. [Accessed 3 August 2025].

[10] S. Brass, "Deductive Databases and Logic Programming (Winter 2007/2008)," 2007. [Online]. Available: https://users.informatik.uni-halle.de/~brass/lp07/c7_magic.pdf. [Accessed 5 August 2025].

[11] K. T. Tekle and Y. A. Liu, "Precise complexity analysis for efficient datalog queries," in *PPDP '10: Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming*, 2010.

[12] Y. Guo, J. Heflin and J. Pan, "Lubm: A benchmark for owl knowledge base systems," *Journal of Web Semantics,* vol. 3, no. 2-3, pp. 158-182, 2005.

[13] D. Allemang and J. Hendler, Allemang, Dean, and James Hendler. Semantic web for the working ontologist: effective modeling in RDFS and OWL, Elsevier, 2011.

[14] S. T. L. A. N. A. S. Cao, "Transactions on Computational Intelligence XIII," in *The web ontology rule language OWL 2 RL+ and its extensions.*, Berlin, Springer Berlin Heidelberg, 2014, pp. 152-175.

[15] I. H. R. V. S. D. B. N. Grosof, "Description logic programs: combining logic programs with description logic," in *The Web Conference*, 2003.

[16] K. M. D. A. Bader, "Gtgraph : A synthetic graph generator suite," Atlanta, GA, 2006.

[17] J. Leskovec and J. Mcauley, "Learning to Discover Social Circles in Ego Networks," *Advances in neural information processing systems,* vol. 25, 2012.

[18] T. Krantz and A. Jonker, "What is ground truth?," IBM, 20 December 2024. [Online]. Available: https://www.ibm.com/think/topics/ground-truth. [Accessed 3 August 2025].

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

Appendix A. Full transformations of linear and nonlinear transitive closure programs.

```
tc_bb("x", "y") <- [magic_tc_bb("x", "y"), e("x", "y")]
tc_bb("x", "y") <- [magic_tc_bb("x", "y"), tc_bb("x", "y"), tc_bf("y",
"z")]
tc_bf("x", "y") <- [magic_tc_bf("x"), e("x", "y")]
tc_bf("x", "y") <- [magic_tc_bf("x"), tc_bf("x", "y"), tc_ff("y", "z")]
tc_ff("x", "y") <- [e("x", "y")]
tc_ff("x", "y") <- [tc_ff("x", "y"), tc_ff("y", "z")]
magic_tc_bf("y") <- [magic_tc_bb("x", "y"), tc_bb("x", "y")]
```

Nonlinear transitive closure program with a (bound, bound) query

```
tc_bf("x", "y") <- [magic_tc_bf("x"), e("x", "y")]
tc_bf("x", "y") <- [magic_tc_bf("x"), tc_bf("x", "y"), tc_ff("y", "z")]
tc_ff("x", "y") <- [e("x", "y")]
tc_ff("x", "y") <- [tc_ff("x", "y"), tc_ff("y", "z")]
```

Nonlinear transitive closure program with a (bound, free) query

```
tc_bb("x", "y") <- [magic_tc_bb("x", "y"), e("x", "y")]
tc_bb("x", "y") <- [magic_tc_bb("x", "y"), e("x", "y"), tc_bf("y", "z")]
tc_bf("x", "y") <- [magic_tc_bf("x"), e("x", "y")]
tc_bf("x", "y") <- [magic_tc_bf("x"), e("x", "y"), tc_bf("y", "z")]
magic_tc_bf("y") <- [magic_tc_bf("x"), e("x", "y")]
magic_tc_bf("y") <- [magic_tc_bb("x", "y"), e("x", "y")]
```

Linear transitive closure program with a (bound, bound) query

```
tc_bf("x", "y") <- [magic_tc_bf("x"), e("x", "y")]
tc_bf("x", "y") <- [magic_tc_bf("x"), e("x", "y"), tc_bf("y", "z")]
magic_tc_bf("y") <- [magic_tc_bf("x"), e("x", "y")]
```

Linear transitive closure program with a (bound, free) query

License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Merlin Kasesalu ,
(author´s name)

1. grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

   Back to Magic ,
   (title of thesis)

   supervised by Bruno Rucy Carneiro Alves de Lima ;
   (supervisor´s name)

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Merlin Kasesalu

10/08/2025