

Extended Magic for Negation: Efficient Demand-Driven Evaluation of Stratified Datalog with Precise Complexity Guarantees

K. Tuncay Tekle Yanhong A. Liu

Department of Computer Science, Stony Brook University

tuncay,liu@cs.stonybrook.edu

Given a set of Datalog rules, facts, and a query, answers to the query can be inferred bottom-up starting from the facts or top-down starting from the query. For efficiency, top-down evaluation is extended with memoization of inferred facts, and bottom-up evaluation is performed after transformations to make rules driven by the demand from the query. Prior work has shown their precise complexity analysis and relationships. However, when Datalog is extended with even stratified negation, which has a simple and universally accepted semantics, transformations to make rules demand-driven may result in non-stratified negation, which has had many complex semantics and evaluation methods.

This paper presents (1) a simple extension to demand transformation, a transformation to make rules demand-driven for Datalog without negation, to support stratified negation, and (2) a simple extension to an optimal bottom-up evaluation method for Datalog with stratified negation, to handle non-stratified negation in the resulting rules. We show that the method provides precise complexity guarantees. It is also optimal in that only facts needed for top-down evaluation of the query are inferred and each firing of a rule to infer such a fact takes worst-case constant time. We extend the precise relationship between top-down evaluation and demand-driven bottom-up evaluation to Datalog with stratified negation. Finally, we show experimental results for performance, as well as applications to previously challenging examples.

1 Introduction

Datalog [19] is a logic language for deductive databases [1], security [11], networking [18], semantic web [9], and many other applications [29].

Given a set of Datalog rules, facts, and a query, answers to the query can be inferred using bottom-up evaluation starting with the facts or top-down evaluation starting with the query. The dominant strategies for efficient evaluations are top-down evaluation with *variant tabling* [26] to memoize and reuse answers to subqueries, and bottom-up evaluation with the *magic set transformation* (MST) [5] so that evaluation of the transformed rules are driven by demand from the query.

The performance of Datalog engines is difficult to understand due to the multitude of factors involved. For example, the performance of different tabling strategies vary drastically [23], and bottom-up evaluation after MST may be much slower than the bottom-up evaluation of the original rules [25]. Choosing the best evaluation method for a given set of rules requires precise complexity analysis of each evaluation method. Recently, *demand transformation* (DT), resulting in simpler rules with better space complexity than MST has been introduced, and the time and space complexities of top-down evaluation with tabling and bottom-up evaluation after DT has been precisely established.

Stratified negation is a simple kind of negation with a simple and universally accepted semantics, and it is handled gracefully by top-down evaluation with tabling requiring almost no change. However, MST

applied to rules with stratified negation is known to result in rules with non-stratified negation, which has many complex semantics, such as well-founded semantics [13] and stable model semantics [14]. Under such semantics, precise complexity analysis is difficult, and worst-case behavior is prohibitive, as in the quadratic time complexity in the size of the grounded rules for well-founded semantics. A more recent semantics, founded semantics [17], can evaluate a certain class of rules with arbitrary negation in linear time, but not for all rules. Other efforts to address this problem have been made, as discussed in Section 7, but they are more complicated than even MST, and none of them provide precise complexity guarantees, or proofs that they match top-down evaluation with tabling.

This paper presents a new method consisting of (1) a simple extension to demand transformation, a transformation to make rules demand-driven for Datalog without negation, to support stratified negation, and (2) a simple extension to an optimal bottom-up evaluation method for Datalog with stratified negation, to handle non-stratified negation in the resulting rules. We show that the method provides precise complexity guarantees, which is worst-case linear time in the size of the grounded rules, in contrast to existing quadratic time results. The method only infers facts needed for top-down evaluation of the query, and each firing of a rule to infer such a fact takes worst-case constant time. We extend the precise relationship between top-down evaluation and demand-driven bottom-up evaluation to Datalog with stratified negation. Finally, we show experimental results for performance, as well as applications to previously challenging examples.

2 Datalog with negation

Datalog [19] is a language for defining rules, facts, and queries, where rules can be used with facts to answer queries.

A Datalog rule is of the form:

$$p(a_1, \dots, a_k) \leftarrow p_1(a_{11}, \dots, a_{1k_1}), \dots, p_h(a_{h1}, \dots, a_{hk_h}).$$

where h is a finite natural number, each p_i (respectively p) is a predicate of finite number k_i (respectively k) arguments, each a_{ij} and a_i is either a constant or a variable, and each variable in the arguments of p must also be in the arguments of some p_i .

If $h = 0$, then each a_i must be a constant, in which case $p(a_1, \dots, a_k)$ is called a *fact*.

For the rest of the paper, *rule* refers only to the case where $h \geq 1$, in which case each $p_i(a_{i1}, \dots, a_{ik_i})$ is called a *hypothesis*, also known as *premise*, and $p(a_1, \dots, a_k)$ is called the *conclusion*.

The meaning of a set of rules and facts is the set of facts that are given or can be inferred using the rules.

A *query* is of the form $p(a_1, \dots, a_k)?$ where each argument a_i is either a constant or a variable. Answers to a query is the set of facts, given or inferred, that match the query, where a fact f *matches* a query q if there is a substitution of the variables in q to constants such that q under the substitution is the same as f .

We call a predicate that appears in the conclusion of a rule an *intensional* predicate, and a predicate that does not appear in the conclusion of any rule but appears only in given facts an *extensional* predicate. We assume without loss of generality that intensional and extensional predicates are distinct.

In examples, we use letters for variables in arguments of predicates, and we use literal numbers for constants.

Example. We use the following rules for transitive closure as a running example.

$$p(x, y) \leftarrow e(x, y). \quad (\text{R1})$$

$$p(x, z) \leftarrow e(x, y), p(y, z). \quad (\text{R2})$$

We consider the query $p(1, x)?$. It asks for all nodes reachable from 1 following the edges. ■

Stratified negation. Datalog can be extended with *stratified negation* by extending rules to be of the form, where [not] indicates that not is optional:

$$p(a_1, \dots, a_k) \leftarrow [\text{not}]p_1(a_{11}, \dots, a_{1k_1}), \dots, [\text{not}]p_h(a_{h1}, \dots, a_{hk_h}).$$

and imposing the constraint that there is no cyclic dependency between any predicate and a negated predicate, i.e., a predicate preceded with a not. For example, replacing (R2) in the running example with the following rule, where q is an extensional predicate, results in Datalog with stratified negation:

$$p(x, z) \leftarrow e(x, y), p(y, z), \text{not } q(x, z). \quad (\text{R2}'\text{S})$$

because there is no cyclic dependency between p and q .

Formally, we say that predicate p_1 (negatively) depends on predicate p_2 if there is a rule whose conclusion's predicate is p_1 and there is a (negative) hypothesis in the rule whose predicate is p_2 . A set of rules is *stratified* if for any two predicates p_1 and p_2 , if p_1 negatively depends on p_2 , then p_2 does not transitively depend on p_1 . Predicates in a stratified set of rules can be split into numbered *strata* such that for any two strata s_1 and s_2 , if a predicate in s_1 negatively depends on a predicate in s_2 , then s_1 has a higher number than s_2 .

Datalog with stratified negation has a simple and universally accepted semantics, where rules having predicates in a lower strata are evaluated first, and negated hypotheses under a substitution is considered false if the corresponding facts have not been inferred. This semantics coincides with all well-known semantics for negation, including perfect model semantics [22], well-founded semantics [13], and stable model semantics [14].

Non-stratified negation. Removing the constraint that there is no cyclic dependency between any predicate and a negated predicate may result in non-stratified negation, which has no universally accepted semantics. Consider the following rule:

$$t(x) \leftarrow \text{not } t(x).$$

For a constant c , if $t(c)$ is false, then it is also true by this rule, therefore standard semantics does not apply. Many sophisticated semantics have been developed, including well-founded semantics [13] and stable model semantics [14]. Also, under sophisticated semantics, precise complexity analysis is difficult, and worst-case behavior is prohibitive, e.g., quadratic time in the size of the grounded rules for well-founded semantics [7]. Therefore, rules with non-stratified negation must be avoided whenever possible.

3 Demand-driven evaluation and challenge of stratified negation

This section describes top-down and bottom-up methods for answering queries, and challenge of stratified negation.

Top-down evaluation with tabling. To answer a query, top-down evaluation starts with the query, generates subqueries from hypotheses of rules whose conclusions match the query, considering rules in the order given, and considering hypotheses from left to right, and does so repeatedly until the subqueries match given facts.

Straightforward top-down evaluation may lead to repeated subqueries, or even infinite recursion for recursive rules such as (R2) in the running example. To address this problem, *tabling* memoizes answers to queries encountered, and reuses the answers when a query is encountered again.

In this paper, we consider top-down evaluation using the dominant tabling strategy, variant tabling without early completion, which exploits all ways to infer the answers to a query modulo variable renaming [10]. We refer to this evaluation as TOP-DOWN in the rest of the paper.

Stratified negation is generally supported for queries that are *non-floundering* [10]. A query is non-floundering with respect to a set of rules if during TOP-DOWN of the query, all subqueries for negated hypotheses have all of their arguments bound.

In this paper, we only consider non-floundering queries of Datalog with stratified negation, because these queries can be evaluated by TOP-DOWN with a trivial extension to test the negation.

Optimal bottom-up evaluation and demand-driven transformations. Bottom-up evaluation starts with given facts, infers new facts from conclusions of rules whose hypotheses match existing facts, and does so repeatedly until no more facts can be inferred.

An optimal method [16] transforms any given set of Datalog rules into an efficient specialized procedural implementation with guaranteed worst-case time and space complexities, and computes the complexities from the rules.

In particular, rules are first transformed to remove singleton variables (variables that occur in only one hypothesis) and multiple occurrences of the same variable in a single hypothesis. Next, rules with more than two hypotheses are decomposed into rules of two hypotheses. Then, the least fixed-point specification of the rules is transformed to a while-loop that considers given and inferred facts incrementally, one at a time, where auxiliary indices are used to find each matching fact in constant time.

The evaluation is optimal in that only combinations of facts that make the hypotheses of a rule simultaneously true are considered, and each such combination, which leads to a *firing* of the rule, is considered once in constant time. The time complexity is precisely the sum of the number of firings over all rules plus the size of extensional predicates for reading given facts.

In this paper, we only consider the decomposition of rules that takes the two leftmost hypotheses into a new rule at a time. We call this *left-optimal bottom-up evaluation*, because the time complexity of evaluation using this decomposition is optimal for the left-to-right ordering of the hypotheses in a rule. We refer to this evaluation as BOTTOM-UP in the rest of the paper.

Note that the optimal bottom-up evaluation in [16] is not limited to the particular decomposition used by BOTTOM-UP. It can perform time and space complexity calculation for all possible decompositions, and select an optimal one trading space for time, including space for intermediate predicates from the decompositions. We consider BOTTOM-UP in this paper in order to establish correspondence with the left-to-right evaluation of hypotheses in TOP-DOWN.

BOTTOM-UP infers all facts possible from the given facts, and thus may infer many facts not needed for answering the query. For efficient evaluation, the query can be used to limit the facts that can be inferred. This is achieved by transforming the rules to be restricted by the query, like in the well-known *magic set transformation (MST)* [5].

Demand transformation (DT) [27] is such a transformation that results in rules with the same time complexity and exponentially better space complexity in program size than MST. It transforms a set of rules and a query into a new set of rules and a fact, which can infer only facts that can be inferred during TOP-DOWN of the original rules. DT and MST make use of predicate annotations for argument binding patterns. DT differs from MST in that given intensional predicates are not annotated in transformed rules, therefore is much simpler, and much less space is used because a fact of a predicate is stored only once, rather than once for each possible annotation of the predicate.

DT has two stages: (I) compute demand patterns for intensional predicates, that is, a set of pairs $\langle p, s \rangle$ indicating that under TOP-DOWN for the given query, there is a subquery for predicate p with pattern s ,

a string of characters for arguments of the subquery, where the i th character is b if the i th argument is a constant (i.e., *bound*), and f if it is a variable (i.e., *free*), and (II) transform given rules and query using the demand patterns computed, in three steps, as follows.

1. For each demand pattern $\langle p, s \rangle$, and each given rule $p(\text{args}) \leftarrow h_1, \dots, h_n$, generate following rule

$$p(\text{args}) \leftarrow d_{p_s}(a_1, \dots, a_k), h_1, \dots, h_n.$$

where args denotes arguments in the conclusion, a_1, \dots, a_k are arguments in args that are bound by s , i.e., that correspond to character b in s .

2. For the given query q of the form $p(\text{args})?$, generate the following fact:

$$d_{p_s}(a_{b1}, \dots, a_{bl}).$$

where s is the pattern for q , and a_{b1}, \dots, a_{bl} are the constant arguments in args .

3. For each rule generated in Step 1, $c \leftarrow h_0, \dots, h_n$, and each h_i whose predicate is an intensional predicate p , generate the following rule:

$$d_{p_s}(a_1, \dots, a_k) \leftarrow h_0, \dots, h_{i-1}.$$

where s is the pattern of h_i , and a_1, \dots, a_k are the bound arguments of h_i .

Example. For the running example, the only demand pattern is $\langle p, bf \rangle$, because the only type of query encountered during TOP-DOWN is one where the first argument of p is a constant. DT yields the following rules and fact:

$$\begin{aligned} p(x, y) &\leftarrow d_{p_bf}(x), e(x, y). & (R1') & \quad (DT \text{ Step } 1) \\ p(x, z) &\leftarrow d_{p_bf}(x), e(x, y), p(y, z). & (R2') & \quad (DT \text{ Step } 1) \\ d_{p_bf}(1). & & (DF) & \quad (DT \text{ Step } 2) \\ d_{p_bf}(y) &\leftarrow d_{p_bf}(x), e(x, y). & (DR) & \quad (DT \text{ Step } 3) \end{aligned}$$

(R1') and (R2') restrict the original (R1) and (R2) using a new demand predicate d_{p_bf} indicating demand on the first of two arguments of predicate p . (DF) is a new fact that corresponds to demand by the given query where the first of the two arguments of p is 1. (DR) propagates demand in (R2') from $d_{p_bf}(x)$ via $e(x, y)$ to demand on the first argument y in $p(y, z)$. ■

Challenge of stratified negation for MST and DT. Not only can TOP-DOWN handle non-floundering queries of stratified Datalog with a trivial extension [10], which checks whether a negative hypothesis is true under the current substitution; but also can BOTTOM-UP be trivially extended and give precise time and space complexity guarantees as for without negation [16], by checking whether a fully instantiated negative hypothesis is true in $O(1)$ time.

It is therefore natural to think that MST and DT would apply to stratified negation as well, adding demands that mimic TOP-DOWN, so that BOTTOM-UP of the resulting rules has the same optimal complexity as for without negation. Unfortunately, this is not true—applying MST to stratified rules, treating demand for a negated hypotheses as demand for the hypothesis without the negation, may result in non-stratified rules [4]. We show that DT has the same problem as MST through the following example.

Example. Consider extending the running example with two additional rules:

$$\begin{aligned} p2(x, y) &\leftarrow \text{not } p(x, y), e2(x, y). & (R3) \\ p2(x, z) &\leftarrow \text{not } p(x, z), e2(x, y), p2(y, z). & (R4) \end{aligned}$$

$p2$ is the transitive closure of $e2$ whose computation does not use any pair in p . The four rules are stratified because only $p2$ depends negatively on p , and p does not transitively depend on $p2$.

Consider the query $p2(1,2)?$. The query is non-floundering because the only demand pattern for the negated hypotheses is $\langle p, bb \rangle$. DT yields the following rules, treating demand for a negated hypothesis as demand for the hypothesis without the negation:

$p(x,y) \leftarrow d_p_bb(x,y), e(x,y).$	(R1')	(DT Step 1)
$p(x,z) \leftarrow d_p_bb(x,z), e(x,y), p(y,z).$	(R2')	(DT Step 1)
$p2(x,y) \leftarrow d_p2_bb(x,y), \text{not } p(x,y), e2(x,y).$	(R3')	(DT Step 1)
$p2(x,z) \leftarrow d_p2_bb(x,z), \text{not } p(x,z), e2(x,y), p2(y,z).$	(R4')	(DT Step 1)
$d_p2_bb(1,2).$	(DF)	(DT Step 2)
$d_p_bb(y,z) \leftarrow d_p_bb(x,z), e(x,y).$	(D1)	(DT Step 3)
$d_p_bb(x,y) \leftarrow d_p2_bb(x,y).$	(D2)	(DT Step 3)
$d_p_bb(x,z) \leftarrow d_p2_bb(x,z).$	(D3)	(DT Step 3)
$d_p2_bb(y,z) \leftarrow d_p2_bb(x,z), \text{not } p(x,z), e2(x,y).$	(D4)	(DT Step 3)

The resulting rules are not stratified: d_p2_bb negatively depends on p , in (D4), and p transitively depends on d_p2_bb through d_p_bb , in (R1') and (D2). ■

How does TOP-DOWN handle stratified negation without changes? The key is that TOP-DOWN has a sequential order for when each subquery is processed, whereas BOTTOM-UP infers facts of demand predicates, i.e., the predicates with prefix $d_$, without a sequential order. We illustrate this for the extended example above.

Example. In the example above, TOP-DOWN would ask $p2(1,2)?$, which asks $\text{not } p(1,2)?$ using rule (R4), and assuming it returns true, TOP-DOWN would find a value for y , say 3 and ask $p2(3,2)?$, then again using (R4) would ask $\text{not } p(3,2)?$, and so on.

In contrast, in the resulting rules from DT, a demand for $p2$ for the third hypothesis of (R4) depends on the truth value of p , whose demand depends on the demand for $p2$ as it appears in a rule whose conclusion's predicate is $p2$, leading to a cyclic definition with negation. ■

It has been shown that well-founded semantics of magic-set transformed stratified rules is two-valued [15]; it can be shown easily that this applies to DT as well. This agrees with the understanding that TOP-DOWN computes well-founded semantics [10]. Then, one could evaluate such rules after DT by computing well-founded semantics. However, the best bottom-up methods known for computing well-founded semantics are quadratic time in the grounded rules [8], which implies a prohibitive $O(k^v \times k^v)$ bound, where k is the number of constants, and v is the maximum number of variables in a rule.

A recent semantics, founded semantics [17] can compute well-founded semantics in linear time in the absence of what they call *closed predicates*. However, it can be shown that the example above requires closed predicates.

A number of serious efforts have been made to address the problem of non-stratified negation in the resulting rules from MST, as discussed in Section 7, but all of them are much more complicated than even MST, and none of them provide precise complexity guarantees, let alone matching TOP-DOWN.

4 Extending demand transformation and bottom-up evaluation for stratified negation

We present simple extensions to DT and BOTTOM-UP so that the demand from the query is precisely captured, and answers to the query are inferred efficiently with precise time complexity guarantees. The resulting evaluation and complexity match TOP-DOWN exactly.

4.1 Extended demand transformation

We extend DT to handle stratified negation by introducing a new predicate $n.p$ for each predicate p that appears in a negated hypothesis and using $n.p$ in place of $\text{not } p$. Predicate $n.p$ therefore stands for the complement of p . We use the demand for $n.p$ to create demand for p . We then extend BOTTOM-UP in the next subsection to infer facts of $n.p$ by exploiting stratification. The extended DT has three steps.

1. Replace each negated hypothesis $\text{not } p(\text{args})$ with $n.p(\text{args})$, where $n.p$ is a new predicate.
 2. Add a rule $n.p(a_1, \dots, a_k) \leftarrow \text{not } p(a_1, \dots, a_k)$. for each $n.p$ of k arguments, where a_1, \dots, a_k are distinct variables.
 3. Apply DT as before, treating demand for $\text{not } p(\text{args})$ as demand for $p(\text{args})$.
- Note that the only rules that contain such negated hypotheses are rules added in Step 2.

Example. For the extended running example, extended DT yields the following rules.

$p(x, y) \leftarrow d_p_bb(x, y), e(x, y)$.	(R1')	(Step 3, DT Step 1)
$p(x, z) \leftarrow d_p_bb(x, z), e(x, y), p(y, z)$.	(R2')	(Step 3, DT Step 1)
$p2(x, y) \leftarrow d_p2_bb(x, y), n.p(x, y), e2(x, y)$.	(R3'')	(Steps 1, 3, DT Step 1)
$p2(x, z) \leftarrow d_p2_bb(x, z), n.p(x, z), e2(x, y), p2(y, z)$.	(R4'')	(Steps 1, 3, DT Step 1)
$n.p(x, y) \leftarrow d_n.p_bb(x, y), \text{not } p(x, y)$.	(N1)	(Steps 2, 3, DT Step 1)
$d_p2_bb(1, 2)$.	(DF)	(Step 3, DT Step 2)
$d_p_bb(y, z) \leftarrow d_p_bb(x, z), e(x, y)$.	(D1)	(Step 3, DT Step 3)
$d_n.p_bb(x, y) \leftarrow d_p2_bb(x, y)$.	(D2')	(Step 3, DT Step 3)
$d_n.p_bb(x, z) \leftarrow d_p2_bb(x, z)$.	(D3')	(Step 3, DT Step 3)
$d_p2_bb(y, z) \leftarrow d_p2_bb(x, z), n.p(x, z), e2(x, y)$.	(D4')	(Steps 1, 3, DT Step 3)
$d_p_bb(x, z) \leftarrow d_n.p_bb(x, z)$.	(DN1)	(Steps 2, 3, DT Step 3)

Note the following observations:

- Except for (N1), the resulting rules contain no negation, because $\text{not } p(\text{args})$ has been replaced with $n.p(\text{args})$ in all other rules.
- The rule (DN1) is added in Step 3 of extended DT for the demand resulting from the second hypothesis of the rule (N1).
- The new predicate $n.p$ is still in a cycle containing negation with p , so the rules are not stratified. ■

Next, we extend bottom-up evaluation to handle resulting rules that contain negation so that facts of new predicates $n.p$, and in turn facts of all predicates, can be inferred correctly.

4.2 Extended bottom-up evaluation

We extend BOTTOM-UP so that facts for each predicate $n.p$ are inferred and the rest of the evaluation proceeds as before with optimal time complexity.

We make the following observation that underlies the idea of the extension.

Lemma 1 *Given a set of stratified rules rs and a query, let rs' be the resulting rules after extended DT. After BOTTOM-UP of rs' , consider predicate p in the lowest stratum in rs among predicates such that $d_n.p_s(\text{args})$ has been inferred: if $p(\text{args})$ has not been inferred, then the generated rule $n.p(a_1, \dots, a_k) \leftarrow d_n.p_s(a_1, \dots, a_k), \text{not } p(a_1, \dots, a_k)$. by extended DT can be used to infer $n.p(\text{args})$.*

Proof. Since p is in the lowest stratum among all predicates for which there is a negated demand, there is no negated hypothesis that p transitively depends on. If $d_n.p_s(\text{args})$ has been inferred, then because of a rule introduced in Step 3 of extended DT, $d_p_s(\text{args})$ has been inferred as well. Since p only depends on d_p_s and other positive hypotheses, $p(\text{args})$ must be inferred by BOTTOM-UP if true,

and not inferred otherwise. Therefore if it has not been inferred, it is false. Since $n.p$ is the complement of p , $n.p(\text{args})$ is true. \square

Therefore, running BOTTOM-UP without rules containing negation, after BOTTOM-UP reaches a fixed point, new facts for a predicate $n.p$ in the lowest stratum, for which the corresponding demand predicate $d_n.p_s$ has a fact, can be inferred. After inferring facts for $n.p$, BOTTOM-UP may be applied again, and reach a new fixed point. After the new fixed point is reached, if there are new facts for a demand predicate $d_n.p_s$, then $n.p$ facts can be inferred again for such facts for the predicate in the lowest stratum, and so on. Precisely, BOTTOM-UP after extended DT is extended as follows, and we refer to it as extended BOTTOM-UP after extended DT.

Repeat the following steps until fixed point.

1. Perform BOTTOM-UP without using rules that infer facts of predicates of the form $n.p$. Such rules are first added in Step 2 of extended DT and then transformed in Step 3 of extended DT, and are of the form $n.p(a_1, \dots, a_k) \leftarrow d_n.p_s(a_1, \dots, a_k), \text{ not } p(a_1, \dots, a_k)$. where s is all b 's.
2. Use a rule to infer a new fact $n.p(\text{args})$ if $d_n.p_s$ is in the lowest stratum among predicates of the form $d_n.q_s$ for some predicate q for which (i) $d_n.p_s(\text{args})$ has been inferred and (ii) $p(\text{args})$ has not been inferred.

We extend Lemma 1 to show correctness at any iteration of the extended evaluation.

Lemma 2 *Suppose $d_n.p_s(\text{args})$ is inferred during extended BOTTOM-UP after extended DT. If $p(\text{args})$ is false for the original set of rules, then $n.p(\text{args})$ is inferred by extended BOTTOM-UP after extended DT, and not inferred otherwise.*

Proof. (Case 1) $p(\text{args})$ is false for the original set of rules. Then, this fact cannot be inferred by the rules defining p in the rules obtained after extended DT since they are more restrictive than the original rules. Given that $d_n.p_s(\text{args})$ (say f) is inferred during extended BOTTOM-UP after extended DT, it is guaranteed that if the rule for $n.p$ is ever used, then $n.p(\text{args})$ will be inferred. The rule for $n.p$ is guaranteed to be used since at some point in the iteration f will be the fact in the lowest stratum in Step 2.

(Case 2) $p(\text{args})$ is true for the original set of rules. Since $d_n.p_s(\text{args})$ is inferred during extended BOTTOM-UP after extended DT, then $d_n.p_s(\text{args})$ is inferred also due to the rule generated in Step 3 of extended DT. Hence, $p(\text{args})$ will be inferred during extended BOTTOM-UP after extended DT since there is a demand for it, and all of the hypotheses that define it must be in a lower stratum, and hence correctly inferred also. Since $p(\text{args})$ is inferred, $n.p(\text{args})$ cannot be inferred. \square

Theorem 1 (Extended BOTTOM-UP after extended DT matches TOP-DOWN) *A fact is inferred during extended BOTTOM-UP of a set of stratified rules and query after extended DT iff it is inferred during TOP-DOWN of the rules and query.*

Proof Sketch. This relationship is already known for BOTTOM-UP after DT and TOP-DOWN for rules without stratified negation, therefore we only need to extend the argument to handle negated hypotheses.

If a demand fact for a negated hypothesis is inferred during extended BOTTOM-UP after extended DT, then all hypotheses to its left must be true, therefore a corresponding query must be evaluated during TOP-DOWN, and vice versa. So demand facts inferred correspond exactly to subqueries encountered during TOP-DOWN. By Lemma 2, if there is a demand for a negated hypothesis, then its corresponding fact is inferred correctly. Therefore, extended BOTTOM-UP after extended DT corresponds exactly to TOP-DOWN in terms of inferred facts. \square

5 Precise complexity guarantees

We give precise time complexity analysis using the following parameters as in [16].

- $\#p$: number of facts of predicate p , called *size of* p .
- $\#p.i_1, \dots, i_n / j_1, \dots, j_m$: maximum number of combinations of values taken by the i_1, \dots, i_n -th arguments of facts of predicate p , given any fixed value of the j_1, \dots, j_m -th arguments.

There are two forms of rules after decomposition of rules. When a rule has one hypothesis, it is of the form: $p(x_1, \dots, x_n) \leftarrow q(y_1, \dots, y_m)$. The number of times this rule can fire is the number of facts of q , therefore the time complexity incurred by this rule is $O(\#q)$. In fact, we can omit the complexity of such rules, because (i) if q is an extensional predicate, then all its facts need to be read in, therefore the complexity is already incurred by the reading of the input, (ii) if q is an intensional predicate, then its size is bound by the complexity of the rules that infer its facts.

When a rule has two hypotheses, it is of the form: $p(\text{args}) \leftarrow q(x_1, \dots, x_n), r(y_1, \dots, y_m)$. To calculate the number of firings, we can first consider facts of q and find matching facts of r such that common variables in the two hypotheses take the same value. Therefore, only variables in the second hypothesis but not in the first can take different values for each fact of q . Let C_{12} be the set of indices j in $1..m$ such that y_j is a common variable in the two hypotheses, then the complexity is bounded by $O(\#q \times \#r.i \notin C_{12} / j \in C_{12})$. Symmetrically, we can consider facts of r and find matching facts of q . Let C_{21} be symmetrically the set of indices j in $1..n$ such that x_j is a common variable, then the complexity is also bounded by $O(\#r \times \#q.i \notin C_{21} / j \in C_{21})$. Both are upper bounds, so the complexity is bounded by the minimum of the two:

$$O(\min(\#q \times \#r.i \notin C_{12} / j \in C_{12}, \#r \times \#q.i \notin C_{21} / j \in C_{21}))$$

Example. For the running example, rule (R1) incurs the time complexity $O(\#e)$, and rule (R2) incurs the time complexity $O(\min(\#e \times \#p.2/1, \#p \times \#e.1/2))$. ■

5.1 Complexity characteristics of extended BOTTOM-UP after extended DT

Theorem 2 (Extensions preserve complexity) *Extended BOTTOM-UP after extended DT preserves the complexity characteristics and optimality of BOTTOM-UP after DT.*

Proof. The only difference in extended BOTTOM-UP is Step 2 where a demand fact for predicates of the form $n.p$ in the lowest stratum is found. Such a fact can be found in constant time in data complexity, by searching for demand facts starting from the lowest stratum upwards. Therefore, the extension preserves the complexity characteristics and the optimality of BOTTOM-UP after DT. □

The precise time and space complexity of TOP-DOWN, and its relationship with BOTTOM-UP after MST and DT, were open until [27]. The results in [27] include the following theorems—they establish that the time complexity of TOP-DOWN and that of BOTTOM-UP after DT are equal for rules with at most two hypotheses each, and for general Datalog rules, BOTTOM-UP after DT is equal to or faster than TOP-DOWN.

Theorem 3 (BOTTOM-UP after DT equals TOP-DOWN on decomposed rules [27]) *Let P be a set of Datalog rules and a query, such that the rules have no singleton variables, and there are no more than two hypotheses per rule. Let P' be the set of rules and fact after demand transformation of P . Let T_{td} be the asymptotic time complexity of TOP-DOWN of P , and T_{bu} be the asymptotic time complexity of BOTTOM-UP of P' . Then, $T_{bu} = T_{td}$.*

Theorem 4 (BOTTOM-UP after DT beats TOP-DOWN [27]) *Let P be a set of Datalog rules and a query. Let P' be the set of rules and fact after demand transformation of P . Let T_{td} be the asymptotic time complexity of TOP-DOWN of P , and T_{bu} be the asymptotic time complexity of BOTTOM-UP of P' . Then, $T_{bu} \leq T_{td}$.*

By Theorem 2, and the fact that TOP-DOWN behaves identically for stratified negation, these theorems can be extended as follows.

Corollary 1 (Extended BOTTOM-UP after extended DT equals TOP-DOWN on decomposed rules)

Let P be a set of stratified rules and a non-floundering query, such that the rules have no singleton variables, and there are no more than two hypotheses per rule. Let P' be the set of rules and fact after extended demand transformation of P . Let T_{td} be the asymptotic time complexity of TOP-DOWN of P , and T_{bu} be the asymptotic time complexity of extended BOTTOM-UP of P' . Then, $T_{bu} = T_{td}$.

Corollary 2 (Extended BOTTOM-UP after extended DT beats TOP-DOWN) *Let P be a set of stratified rules and a non-floundering query. Let P' be the set of rules and fact after extended demand transformation of P . Let T_{td} be the asymptotic time complexity of TOP-DOWN of P , and T_{bu} be the asymptotic time complexity of extended BOTTOM-UP of P' . Then, $T_{bu} \leq T_{td}$.*

Example. For the extended running example after extended DT, we calculate the time complexity by considering the most complex rule, copied below.

$$p2(x,z) \leftarrow d_p2_bb(x,z), n.p(x,z), e2(x,y), p2(y,z). \quad (R4'')$$

Extended BOTTOM-UP decomposes this rule into three rules:

$$i1(x,z) \leftarrow d_p2_bb(x,z), n.p(x,z). \quad (R4''1)$$

$$i2(x,y,z) \leftarrow i1(x,z), e2(x,y). \quad (R4''2)$$

$$p2(x,z) \leftarrow i2(x,y,z), p2(y,z). \quad (R4''3)$$

Rules (R4''1) and (R4''3) have a hypothesis containing all variables occurring in the rule, so they do not contribute any extra complexity. The time complexity incurred by (R4''2) is

$$O(\min(\#i1 \times \#e2.2/1, \#e2 \times \#i1.1/2)).$$

Note that this complexity is precise, and can be very small because $i1$ is the intersection of d_p2_bb and $n.p$.

The complexity benefit of extended BOTTOM-UP after extended DT is even more apparent when contrasted to evaluating the resulting rules bottom-up for well-founded semantics whose worst-case time complexity is $O(k^6)$ where k is the number of constants, whereas the worst-case time complexity using our method is $O(k^3)$. ■

6 Applications and experiments

6.1 Experiments on the extended running example

To confirm the effectiveness of our method, we implemented our method to perform extended DT, and then created a Python program that performs extended BOTTOM-UP on the resulting rules. We compare our results with state-of-the-art systems: `clingo` [12], an ASP solver, `DLV2`, the latest version of a deductive database with ASP and dynamic magic sets [3, 2], and `XSB` [10], the dominant engine with tabling and well-founded semantics.

We show experiments on the extended running example. For clingo, which does not support answering queries, we give it the transformed set of rules and facts. For XSB, we used a special construct, `load_dynca` to turn off indexing when loading facts, improving the performance more than 10 times. We use facts generated for predicates `e` and `e2`, by varying the number of nodes and edges. The running times were captured on an Intel Core i5 2.8 GHz with 8 GB RAM, with PyPy 7.1.0 to run Python programs, clingo 5.3.0, DLV2, and XSB 3.8, including the time for reading facts and producing output, and are averaged over 5 runs. The results are shown in Table 6.1, where K stands for 1000.

Table 1: Running times in secs comparing our extended BOTTOM-UP (EBU), clingo, DLV2, XSB

# Nodes	# Edges	EBU	clingo	clingo/EBU	DLV2	DLV2/EBU	XSB	XSB/EBU
1K	200K	2.392	5.549	2.31	1.336	0.56	1.201	0.50
1K	400K	4.597	9.845	2.14	2.599	0.57	2.471	0.54
1K	600K	6.898	13.783	1.99	4.972	0.72	3.606	0.52
2K	600K	7.871	18.129	2.30	5.573	0.71	3.676	0.47
2K	800K	10.999	24.059	2.18	8.012	0.73	4.731	0.43
2K	1000K	13.978	29.669	2.12	7.260	0.52	6.008	0.43

One can see that the running times for extended BOTTOM-UP are tightly coupled with the number of edges as shown in the formula in Example 5.1. It is difficult to speculate on the time and space complexities for clingo and DLV2 because precise complexity analysis as shown for our method is not known. The strength of our method can be seen in contrast to these mature systems even though our Python program is not optimized for constant factors, and Python is known to be much slower than C and C++ used to implement XSB, DLV2, and Clingo.

6.2 Balbin's not-reach-in-reach2 problem

Balbin et al. [4, Example 12] give the following rules to show the challenge of non-stratified negation resulting from applying MST on stratified rules (some predicates/variables have been renamed for readability):

$$r(x) \leftarrow s(x). \quad (\text{B1})$$

$$r(x) \leftarrow e(x,y), r(y). \quad (\text{B2})$$

$$r2(x) \leftarrow s2(x). \quad (\text{B3})$$

$$r2(x) \leftarrow \text{not } r(x), e2(x,y), r2(y). \quad (\text{B4})$$

along with the query $r2(1)?$.

Using our method, the resulting rules from extended DT are:

$$r(x) \leftarrow d_r_b(x), s(x). \quad (\text{B1}')$$

$$r(x) \leftarrow d_r_b(x), e(x,y), r(y). \quad (\text{B2}')$$

$$r2(x) \leftarrow d_r2_b(x), s2(x). \quad (\text{B3}')$$

$$r2(x) \leftarrow d_r2_b(x), n.r(x), e2(x,y), r2(y). \quad (\text{B4}')$$

$$n.r(x) \leftarrow d_n.r_b(x), \text{not } r(x). \quad (\text{N1})$$

$$d_r2_b(1). \quad (\text{DF})$$

$$d_r_b(y) \leftarrow d_r_b(x), e(x,y). \quad (\text{D1})$$

$$d_n.r_b(y) \leftarrow d_r2_b(x). \quad (\text{D2})$$

$$d_r2_b(y) \leftarrow d_r2_b(x), n.r(x), e2(x,y). \quad (\text{D3})$$

$$d_r_b(x) \leftarrow d_n.r_b(x). \quad (\text{D4})$$

These rules can be evaluated in linear time in the input size, because after decomposition into rules with two hypotheses, each rule has at least one hypothesis with extensional predicate `s`, `e`, `s2`, `e2` that contains all variables in the rule. Contrast this to bottom-up evaluation for well-founded semantics, which would have only the bound of $O(k^4)$, where k is the number of constants, since there is a rule with two variables.

6.3 Meskes-Noack's no-extra-joins-in-path problem

Meskes and Noack [20, Example 1] give the following rules to show that a negated predicate does not need to be in a positive cycle for non-stratified rules to occur after MST (some predicates/variables have been renamed for simplicity):

$$s(x) \leftarrow q(x,z), r(z,y). \quad (M1)$$

$$p(x,y) \leftarrow e(x,y), \text{not } s(y). \quad (M2)$$

$$p(x,z) \leftarrow e(x,y), p(y,z), \text{not } s(y). \quad (M3)$$

along with the query $p(1,y)?$. It can be seen that the only negated predicate is s , and it is simply defined using extensional predicates, but after DT, non-stratified rules are obtained.

Extended DT yields the following rules:

$$s(x) \leftarrow d_s_b(x), q(x,z), r(z,y). \quad (M1')$$

$$p(x,y) \leftarrow d_p_bf(x), e(x,y), n.s(y). \quad (M2')$$

$$p(x,z) \leftarrow d_p_bf(x), e(x,y), p(y,z), n.s(y). \quad (M3')$$

$$n.s(x) \leftarrow d_n.s_b(x), \text{not } s(x). \quad (N1)$$

$$d_p_bf(1). \quad (DF)$$

$$d_n.s_b(y) \leftarrow e(x,y). \quad (D1)$$

$$d_p_bf(y) \leftarrow d_p_bf(x), e(x,y). \quad (D2)$$

$$d_n.s_b(y) \leftarrow d_p_bf(x), e(x,y), p(y,z). \quad (D3)$$

$$d_s_b(x) \leftarrow d_n.s_b(x). \quad (D4)$$

(M3') after decomposition gives the highest time complexity: $O(\min(\#i \times \#p.2/1, \#p \times \#i.1/2))$ where i is an intermediate predicate for the leftmost two hypothesis, and is a subset of e restricted by d_p_bf for the first argument. Like our original running example, the time complexity is bounded by $O(k^3)$ using extended BOTTOM-UP after extended DT versus $O(k^6)$ for bottom-up evaluation for well-founded semantics.

7 Related work and conclusion

Datalog has been extensively studied, especially methods for efficient evaluation [19, 1].

Top-down evaluation with variant tabling was introduced in the 1980s [26] and has been widely studied as well. It has been implemented in top-down evaluation engines such as XSB [10] and SWI [30]. Optimal bottom-up evaluation was given in [16]. Incremental maintenance of Datalog with stratified negation was studied in [21].

Transformations for demand-driven bottom-up evaluation have been studied, in many forms of the well-known magic-set transformation (MST) [5], as well as the more recent demand transformation (DT) [27] that our work extends.

A method for precise calculation of time and space complexities of top-down evaluation with variant tabling for Datalog, as well as theorems establishing the precise relationship between top-down and bottom-up evaluations are given in [27, 28].

Although top-down evaluation handles stratified negation without changes, an analogous demand-driven bottom-up evaluation has been challenging to obtain.

Balbin et al. [4] introduce a transformation to a set of magic-set transformed rules, that works for an unidentified class of rules, and does not apply to all rules with stratified negation. They then introduce a new method for bottom-up evaluation with no complexity characterization.

Meskes and Noack [20] extend generalized supplementary MST [6] by ignoring negated hypotheses for rules that infer demand facts to keep the rules stratified, but the method overestimates the demand and hence does not correspond to top-down evaluation.

Ross [24] presents a new transformation whose output is not Datalog, but a higher-order logic, complicating their evaluation and giving no complexity analysis.

Alviano et al. [3] extends magic sets for disjunctive Datalog, but its bottom-up computation requires grounding and stable model search, not giving precise complexity guarantees as we do.

It has been shown that the well-founded semantics of rules obtained by MST for stratified rules is two-valued [15], but the best known complexity for bottom-up evaluation for well-founded semantics is a highly prohibitive quadratic time in the size of the grounded rules.

Our work extends demand transformation and optimal bottom-up evaluation and gives precise complexity guarantees for efficient demand-driven bottom-up evaluation of stratified rules, preserving optimality, and relationships with top-down evaluation.

Future work includes methods for efficient bottom-up evaluation and demand-driven evaluation for non-stratified negation with precise complexity guarantees.

Acknowledgments. This work was supported in part by NSF under grants CCF-1414078 and IIS-1447549.

References

- [1] Serge Abiteboul, Richard Hull & Victor Vianu (1995): *Foundations of Databases*. Addison-Wesley.
- [2] Mario Alviano, Francesco Calimeri, Carmine Dodaro, Davide Fuscà, Nicola Leone, Simona Perri, Francesco Ricca, Pierfrancesco Veltri & Jessica Zangari (2017): *The ASP system DLV2*. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*, Springer, pp. 215–221, doi:10.1007/978-3-319-61660-5_19.
- [3] Mario Alviano, Wolfgang Faber, Gianluigi Greco & Nicola Leone (2012): *Magic sets for disjunctive Datalog programs*. *Artificial Intelligence* 187, pp. 156–192, doi:10.1016/j.artint.2012.04.008.
- [4] Isaac Balbin, Graeme S. Port, Kotagiri Ramamohanarao & Krishnamurthy Meenakshi (1991): *Efficient Bottom-Up Computation of Queries on Stratified Databases*. *J. Log. Program.* 11(3&4), pp. 295–344, doi:10.1016/0743-1066(91)90030-S.
- [5] François Bancilhon, David Maier, Yehoshua Sagiv & Jeffrey D. Ullman (1986): *Magic Sets and Other Strange Ways to Implement Logic Programs*. In: *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 24-26, 1986, Cambridge, Massachusetts, USA*, pp. 1–15, doi:10.1145/6012.15399.
- [6] Catriel Beeri & Raghu Ramakrishnan (1991): *On the Power of Magic*. *J. Log. Program.* 10(3&4), pp. 255–299, doi:10.1016/0743-1066(91)90038-Q.
- [7] K. Berman, J. Schlipf & J. Franco (1995): *Computing the well-founded semantics faster*. In: *Proceedings of the 3rd International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 113–126, doi:10.1007/3-540-59487-6_9.
- [8] Stefan Brass, Jürgen Dix, Burkhard Freitag & Ulrich Zukowski (2001): *Transformation-based bottom-up computation of the well-founded model*. *TPLP* 1(5), pp. 497–538, doi:10.1017/S147106840100103X.
- [9] Andrea Calì, Georg Gottlob & Thomas Lukasiewicz (2012): *A general Datalog-based framework for tractable query answering over ontologies*. *J. Web Sem.* 14, pp. 57–83, doi:10.1016/j.websem.2012.03.001.
- [10] Weidong Chen & David Scott Warren (1996): *Tabled Evaluation With Delaying for General Logic Programs*. *J. ACM* 43(1), pp. 20–74, doi:10.1145/227595.227597.
- [11] John DeTreville (2002): *Binder, a Logic-Based Security Language*. In: *Proc. of the 2002 IEEE Symp. on Security and Privacy (S&P)*, pp. 105–113, doi:10.1109/SECPRI.2002.1004365.

- [12] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub & Marius Thomas Schneider (2011): *Potassco: The Potsdam Answer Set Solving Collection*. *AI Commun.* 24(2), pp. 107–124, doi:10.3233/AIC-2011-0491.
- [13] Allen Van Gelder, Kenneth A. Ross & John S. Schlipf (1991): *The Well-Founded Semantics for General Logic Programs*. *J. ACM* 38(3), pp. 620–650, doi:10.1145/116825.116838.
- [14] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In: *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, pp. 1070–1080.
- [15] David B. Kemp, Divesh Srivastava & Peter J. Stuckey (1995): *Bottom-Up Evaluation and Query Optimization of Well-Founded Models*. *Theor. Comput. Sci.* 146(1&2), pp. 145–184, doi:10.1016/0304-3975(94)00153-A.
- [16] Yanhong A. Liu & Scott D. Stoller (2009): *From Datalog rules to efficient programs with time and space guarantees*. *ACM Trans. Program. Lang. Syst.* 31(6), doi:10.1145/1552309.1552311.
- [17] Yanhong A. Liu & Scott D. Stoller (2018): *Founded Semantics and Constraint Semantics of Logic Rules*. In: *Proc. of the 2018 Intl. Symp. on Logical Foundations of Computer Science*, Springer, pp. 221–241, doi:10.1007/978-3-319-72056-2_14.
- [18] Boon Thau Loo et al. (2009): *Declarative networking*. *Commun. ACM* 52(11), pp. 87–95, doi:10.1145/1592761.1592785.
- [19] David Maier, K. Tuncay Tekle, Michael Kifer & David S. Warren (2018): *Declarative Logic Programming*. chapter Datalog: Concepts, History, and Outlook, ACM, Morgan & Claypool, New York, NY, pp. 3–100, doi:10.1145/3191315.3191317.
- [20] Michael Meskes & Jörg Noack (1993): *The Generalized Supplementary Magic-Sets Transformation for Stratified Datalog*. *Inf. Process. Lett.* 47(1), pp. 31–41, doi:10.1016/0020-0190(93)90154-2.
- [21] Boris Motik, Yavor Nenov, Robert Piro & Ian Horrocks (2019): *Maintenance of datalog materialisations revisited*. *Artificial Intelligence* 269, pp. 76–136, doi:10.1016/j.artint.2018.12.004.
- [22] Teodor C. Przymusiński (1989): *On the Declarative and Procedural Semantics of Logic Programs*. *J. Autom. Reasoning* 5(2), pp. 167–205, doi:10.1007/BF00243002.
- [23] Prasad Rao, C. R. Ramakrishnan & I. V. Ramakrishnan (1996): *A Thread in Time Saves Tabling Time*. In: *JICSLP*, pp. 112–126.
- [24] Kenneth A. Ross (1994): *Modular Stratification and Magic Sets for Datalog Programs with Negation*. *J. ACM* 41(6), pp. 1216–1266, doi:10.1145/195613.195646.
- [25] Damien Sereni, Pavel Avgustinov & Oege de Moor (2008): *Adding magic to an optimising datalog compiler*. In: *SIGMOD Conference*, pp. 553–566, doi:10.1145/1376616.1376673.
- [26] Hisao Tamaki & Taisuke Sato (1986): *OLD Resolution with Tabulation*. In: *Proc. of the 3rd Intl. Conf. on Logic Programming (ICLP)*, pp. 84–98, doi:10.1007/3-540-16492-8_66.
- [27] K. Tuncay Tekle & Yanhong A. Liu (2010): *Precise complexity analysis for efficient datalog queries*. In: *Proceedings of the 12th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 26-28, 2010, Hagenberg, Austria*, pp. 35–44, doi:10.1145/1836089.1836094.
- [28] K. Tuncay Tekle & Yanhong A. Liu (2011): *More efficient datalog queries: subsumptive tabling beats magic sets*. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pp. 661–672, doi:10.1145/1989323.1989393.
- [29] David S. Warren & Yanhong A. Liu (2017): *AppLP: A Dialogue on Applications of Logic Programming*. *Computing Research Repository* arXiv:1704.02375 [cs.PL].
- [30] Jan Wielemaker, Tom Schrijvers, Markus Triska & Torbjörn Lager (2012): *SWI-Prolog. TPLP* 12(1-2), pp. 67–96, doi:10.1017/S1471068411000494.