Backward/Forward with Marking for Update Streams

Moritz Illich, Birte Glimm

Ulm University, James-Franck-Ring, 89081 Ulm, Germany

Abstract

Extending a set of facts with every fact that can be derived based on a set of rules is called materialization. Incremental approaches, like Delete/Rederive (DRed) and Backward/Forward (B/F), allow for efficient adaptations of materialized datasets whenever the original set of facts changes due to an update. To effectively deal with streams of updates, we previously extended DRed with *marking*, where we directly take a look at the next available update in the stream and utilize this insight to prevent repeated rule applications. In this work, we apply this idea on B/F by using marks to indicate and find facts that are deleted by the next update, which enables us to determine facts that need to be checked for some alternative derivation without considering rules for that. An evaluation with both synthetic and real test data demonstrates the marking approach's potential to reduce processing time compared to classical B/F.

Keywords

materialization, incremental reasoning, stream processing

1. Introduction

A possible way to accelerate query answering for description logic ontologies with large ABoxes is the transformation into Datalog [1], where we can utilize optimized reasoners, like RDFox [2], to improve the performance [3]. Answering queries in Datalog is often facilitated by computing the so-called *materialization*, where we extend a set of facts with all its entailed facts based on a set of rules, so that every implicit derivable fact is directly accessible. Materialization from scratch can be time-consuming, which is why we typically use *incremental materialization maintenance* algorithms, like *Delete/Rederive* (DRed) [4] or *Backward/Forward* (B/F) [5], to efficiently deal with updates to a materialized dataset.

While such incremental algorithms only handle one update at a time, *DRed with Marking* [6] presents an extension that directly works with a whole *stream* of updates. When we process an update with this *marking approach*, we already take a look at the next available update from the stream and *mark* facts in the dataset that are affected by a deletion or insertion of the next update. If such a marked fact is involved in a current rule application, we can infer that the derived fact is affected by the next update too and has to be marked as well. This way, we can reduce the number of rule applications that are usually necessary to process the next update, since the markings already provide us with some facts that are relevant for our computations, and, thus, improve the performance of our algorithm.

In this work, we extend B/F with marking and show its potential to reduce processing time compared to classical B/F. The remaining parts start with defining Datalog and materialization maintenance in Section 2, before we discuss related work in Section 3. The formalization of B/F with Marking is in Section 4, followed by an evaluation in Section 5. We conclude in Section 6.

2. Basics and Preliminaries

As done for DRed with Marking [6], we formally define *Datalog* [1] based on countable, disjoint sets of *predicates*, *constants*, and *variables*. A *term* is a constant or a variable. An *atom* has the form $p(t_1, \ldots, t_k)$, where p is a k-ary predicate and each t_i, $1 \leq i \leq k$, is a term. If an atom does not contain variables, it is *ground*. A *fact* is a ground atom and a *dataset* is a finite set of facts. A Datalog *rule* r is a logical implication of the form $B_1, \ldots, B_k \rightarrow H$ where $B_1, \ldots, B_k$ are called *body atoms*, and H is a *head*

DL 2025: 38th International Workshop on Description Logics, September 3–6, 2025, Opole, Poland

moritz.illich@uni-ulm.de (M. Illich); birte.glimm@uni-ulm.de (B. Glimm)

atom. We use $\mathsf{body}(r)$ and $\mathsf{head}(r)$ to denote the set of body atoms and the head atom of r, respectively. A rule is *safe* if variables that appear in the head also appear in a body atom. A *Datalog program* is a finite set of safe rules. Predicates that occur in the head of a rule are called *intensional* (IDB) predicates; all other predicates are *extensional* (EDB).

A *substitution* σ is a partial mapping from variables to constants. For α a term, an atom, a rule, or a set of rules, $\alpha\sigma$ is the result of replacing each occurrence of a variable x in α on which σ is defined with $\sigma(x)$. If r is a rule and σ is a substitution mapping all variables of r to constants, then $r\sigma$ is an *instance* of r. We say that a set of facts S *instantiates* a rule r if there exists a substitution σ such that $\mathsf{body}(r)\sigma = S$. Given a Datalog program P and a dataset I, we define $\rho(P, I) = \{r\sigma \mid r \in P, S \subseteq I, \mathsf{body}(r)\sigma = S\}$ as the set of all rule instances that can be created by instantiating rules in P with subsets of I. A fact is called *derivable* if it appears as head in a rule instance of $\rho(P, I)$. For a program P, we define $P(I) = \bigcup_{r \in \rho(P,I)} \{\mathsf{head}(r)\}$.

2.1. Materialization Maintenance

Let E be a finite dataset of explicit facts. Then, let $I_0 = E$; for each $i \geq 1$, let $I_i = I_{i-1} \cup P(I_{i-1})$, and let $I_n = I_{n+1}$ for some $n \geq 1$. The set I_n is the *materialization* of P w.r.t. E, denoted as $\mathsf{mat}(P, E)$. Let E^- and E^+ be finite datasets with $E^- \subseteq E$, $E \cap E^+ = \emptyset$, and $E^+ \cup E^- \neq \emptyset$. The tuple $U = (E^-, E^+)$ is called an *update* for E, where E^- denotes the facts explicitly deleted from E, and E^+ the facts explicitly added to E, respectively. Applying the update U on E leads to the updated dataset $E' = (E \setminus E^-) \cup E^+$. We allow only EDB predicates in updates. Accordingly, a fact is *implicit* if it has an IDB predicate, and *explicit* if it has an EDB predicate. This is w.l.o.g. as we can replace a k-ary IDB predicate p that is to be used in an update by a new k-ary predicate p' and add rules of the form $p(t_1, \ldots, t_k) \rightarrow p'(t_1, \ldots, t_k)$ such that p becomes an EDB predicate (see, e.g., [7]).

Materialization maintenance is the task of computing the updated materialization $\mathsf{mat}(P, (E_0 \setminus E^-) \cup E^+)$ for a given materialization $\mathsf{mat}(P, E_0)$ and an update $U = (E^-, E^+)$. Let $\hat{U} = \langle U_1, U_2, \ldots \rangle$, with $|\hat{U}| \geq 1$, denote a *stream of updates* for a dataset E_0, where for each $U_i = (E_i^-, E_i^+) \in \hat{U}$, we have $E_i^- \subseteq E_{i-1}$ and $E_{i-1} \cap E_i^+ = \emptyset$, resulting in $E_i = (E_{i-1} \setminus E_i^-) \cup E_i^+$. For a stream of updates $\hat{U}$, materialization maintenance leads to a stream of materialized datasets $\langle \mathsf{mat}(P, E_1), \mathsf{mat}(P, E_2), \ldots \rangle$ where each E_i corresponds to an $U_i \in \hat{U}$.

3. Related Work

As we combine the main idea of Delete/Rederive with Marking [6] with Backward/Forward [5], these algorithms serve as main references for our work and are described in more detail below. In addition, other improvements to DRed and B/F as well as to the well-known counting approach [4] are presented by Motik et al. [8], while Hu et al. [9] describe combined algorithms. Related to the processing of streams, Terdjimi et al. [10] present a tag-based approach that allows for fast re-insertions of deleted facts without repeating rule applications, while Ren and Pan [11] describe a truth maintenance system to handle update streams for $\mathcal{EL}^{++}$ ontologies. DynamiTE [12] performs materialization in parallel for RDF streams and IMaRS [13, 14] utilizes window-based expiration times for efficient incremental adaptations. Unlike Delete/Rederive with Marking, however, all of these approaches only consider one update at a time.

3.1. Delete/Rederive with Marking

DRed with Marking [6] extends the classical *Delete/Rederive* algorithm [4] in order to more efficiently deal with streams of updates, where changes to the materialized dataset might appear faster than they can be processed. The general procedure is still as in the classical algorithm, consisting of the three sequential overdeletion, rederivation, and insertion phases: Given a materialized dataset $I = \mathsf{mat}(P, E)$ and an update $U = (E^-, E^+)$, the *overdeletion* phase removes every fact from I that occurs in E^- or that can be derived by means of such a deleted fact. Even though this procedure ensures that we delete

every fact that is not derivable anymore in the updated materialization, it might also falsely remove facts which still have some alternative derivation that is not affected by any deletion. This problem is solved in the subsequent *rederivation* phase, where we re-add all facts that can still be derived based on the remaining facts. After that, we insert E^+ and compute every new derivable fact in the *insertion* phase to obtain a correctly updated materialization.

Classically, DRed only focuses on one update during its processing. The expanded algorithm, on the other hand, also tries to integrate the next update of the stream into the current processing as soon as it is available. This is achieved by *marking* facts in the dataset that are changed by the next update. In particular, a fact is marked negatively if it is deleted by the next update, whereas a currently deleted fact is marked positively if it is re-added. When such a marked fact is involved in some rule application, then we also mark the derived implicit fact if certain conditions hold. This way, we are able to directly determine some of the fact changes related to the next update and, thus, reduce the number of rule applications that would usually be necessary to process the next update. Concretely, we avoid cases where a rule instance has to be considered again for the next update as illustrated in the following example:

Example 1. *Assume we have a rule $p_1(x), p_2(x) \rightarrow q(x)$ and two consecutive updates $U_a = (\emptyset, \{p_2(c)\})$ and $U_b = (\{p_1(c)\}, \emptyset)$, which we sequentially apply on a dataset $I = \{p_1(c)\}$. When we process U_a, we directly take a look at the next update U_b and (negatively) mark the fact $p_1(c)$ in I, since it will be deleted by U_b. After adding $p_2(c)$ to I, we can apply the rule to derive $q(c)$, which will be marked too, because its derivation depends on $p_1(c)$ that is deleted by the next update, as indicated by the mark. Once we finish the processing of U_a and move on to U_b, we can directly see that $q(c)$ is affected by some deletion due to its mark, without the need to apply the rule again.*

3.2. Backward/Forward

The *Backward/Forward* algorithm [5] was originally introduced as a way to deal with the inherent inefficiency of DRed where falsely deleted facts have to be rederived. While the computation of new derivable facts due to insertions is done as in DRed, the B/F algorithm directly checks if a deleted fact has an alternative derivation, which does not include any deleted facts, to prevent a potential (over)deletion along with the needed rederivation of its consequences.

To find an alternative derivation for some fact F, first, *backward* chaining is applied to determine facts that would be needed to derive F. In detail, we look for rule instances that have F as head and then recursively check if we can "prove" the facts in the rule instance's body, where a fact is considered to be "proven" if it either is explicit (and not deleted) or can be derived entirely by proven facts. To ensure termination in the presence of recursive derivation cycles, each fact is only checked once during backward chaining. For the proving of implicit facts, we perform *forward* chaining by applying rules on already proven facts. If a fact has been checked during backward chaining and was also derived during forward chaining, then it is proven and we keep it, otherwise it has to be deleted.

Example 2. *Assume we have three rules $p_1(x), p_2(x) \rightarrow q(x)$, $p_3(x) \rightarrow q(x)$, and $q(x) \rightarrow r(x)$, and an update $U = (\{p_1(c)\}, \emptyset)$, which we apply on a materialized dataset $I = \{p_1(c), p_2(c), p_3(c), q(c), r(c)\}$. Since $q(c)$ can be derived by the deleted $p_1(c)$ based on the first rule, we check if it has an alternative derivation. Using backward chaining, we match $q(c)$ to the head of the second rule, which then tells us to check $p_3(c)$ in the rule's body next. Since $p_3(c)$ is explicit (as it cannot be derived by any rule), it is directly "proven" and, hence, can be applied for forward chaining with the second rule to derive and, thus, "prove" the previously checked $q(c)$ too. Accordingly, $q(c)$ (and its consequence $r(c)$) must not be deleted.*

The reason why this combination of backward and forward chaining often performs better than DRed is that instead of traversing (and deleting) the facts which can be derived based on a checked fact ($\{r(c)\}$ in Example 2), B/F goes through the facts that potentially support the derivation of the checked fact ($\{p_3(c)\}$ in Example 2), where the latter set of facts is usually smaller. Nevertheless, there still exist cases where B/F can be slower than DRed, for example, when alternative derivations and, thus, rederivations are rare.

4. B/F with Marking

The general idea of the marking approach is that we first mark explicit facts in the dataset which are changed by the next update, and then pass these marks on to implicit facts during rule applications in order to determine further changes that are relevant for the next update. The evaluation of DRed with Marking [6] showed that positive marks, i.e., cases where a currently deleted fact is re-added by the next update, are quite rare and, hence, do not notably contribute to the performance gain of the marking approach. Therefore, we only focus on negative marks for B/F, i.e., situations where facts are deleted by the next update.

An implicit fact is affected by some deletion if there exists a rule instance that has the implicit fact as head and contains a deleted fact in its body. Accordingly, if we have a rule instance that contains at least one marked fact in its body, we may simply mark the head too, as already illustrated in Example 1. In DRed, the overdeletion phase removes every fact that is affected by some deletion, which is why we can also use implicit facts there to propagate marks to other derived facts. In contrast to this, deletions in B/F are accurate, in the sense that we only remove a fact from the dataset when we are sure that it cannot be derived anymore. Accordingly, a marked implicit fact should only be used to mark another fact if we can guarantee that the former cannot be derived anymore due to the next update. As this knowledge can generally not be obtained before actually processing the next update, we may only use marked explicit facts, for which we definitely know that they will be deleted, to mark other facts in B/F.

The computation of marked implicit facts can be done for B/F both during the forward chaining process where we try to prove that a fact is still derivable, and when we determine new derivable facts based on insertions. By looking for marked implicit facts once the processing of the current update is completed, we can directly obtain some facts that have to be checked for alternative derivations when the next update is processed, without the need to first search for appropriate rule instances that tell us that those facts are affected by some deletion.

With the inclusion of the next update in our processing, one might think that we are also able to re-use some of the "proven" facts, and thus avoid repeated backward/forward chaining, by excluding marked proved facts for which we know that they are not derivable in the next update. However, this does generally not work, since markings do usually not cover every fact that has to be deleted for the next update: on the one hand, because we do not allow implicit facts to pass on marks, and on the other hand, since updates might be introduced with a delay, so that some proven facts might not be marked even though they are affected by a deletion.

4.1. Algorithm

The formal description of Backward/Forward extended with marking is presented in Algorithm 1. Here, we use three functions GETNEXT($\hat{U}$), which returns the next update in the stream $\hat{U}$ if one is available, GETLAST(S), which returns the most recently added fact from the set S, and GETMARKED(S), which returns the set of marked facts from S. Additionally, we assume two procedures MARK(S) and UMMARK(S), which add and remove marks for the facts in the given set S, respectively. The input of Algorithm 1 consists of a Datalog program P, a (possibly empty) materialized dataset $I = \mathsf{mat}(P, E_0)$, and a stream of updates $\hat{U} = \langle U_1 = (E_1^-, E_1^+), U_2 = (E_2^-, E_2^+), ... \rangle$. The algorithm's output is a stream of materialized datasets $\langle \mathsf{mat}(P, E_1), \mathsf{mat}(P, E_2), ... \rangle$ with $E_i = (E_{i-1} \setminus E_i^-) \cup E_i^+$ for $i \geq 1$, where each dataset refers to an update in $\hat{U}$. Note that we only work with one program P, i.e., we do not deal with changes to the set of rules. The algorithm's correctness is shown in Appendix A.

One difference to the original B/F algorithm is that we conduct our computations in a big loop to allow the continuous processing of a whole stream of updates. At the beginning of each iteration, we check if there is a current update U_1 to be processed, as well as a next update U_2 that occurs directly after U_1 in the stream. As in DRed with Marking, we consider at most two updates at the same time to keep the marking computations simple. In general, an immediate access to updates is not always possible due to delays in the stream. If we do not have a current update U_1, we wait until one is available (cf. line 5).

Algorithm 1 B/F with Marking

Input: Datalog program P, materialized dataset I, stream of updates $\hat{U}$
Output: stream of materialized datasets

1: $U_1 = (E_1^-, E_1^+) \leftarrow$ null; $U_2 = (E_2^-, E_2^+) \leftarrow$ null; phase $\leftarrow 1$;
2: $C \leftarrow \emptyset$; $D \leftarrow \emptyset$; $V \leftarrow \emptyset$; $W \leftarrow \emptyset$; $Y \leftarrow \emptyset$
3: **repeat**
4: **if** U_1 = null **then**
5: **repeat** $U_1 \leftarrow$ getNext($\hat{U}$) **until** $U_1 \neq$ null
6: $D \leftarrow E_1^-$
7: **if** U_2 = null **then**
8: $U_2 \leftarrow$ getNext($\hat{U}$)
9: **if** $U_2 \neq$ null **then** mark($(I \cup E_1^+) \cap E_2^-$)
10: **if** phase = 1 **then** ▷ DELETIONS
▷ a) Forward chaining with proven facts
11: **if** $\exists r \in \rho(P, I)$: body$(r) \subseteq V$ and head$(r) \notin V \cup Y$ **then**
12: **if** $\exists F \in$ body(r) : F is explicit and marked **then** mark({head(r)})
13: **if** head$(r) \in W$ **then**
14: $V \leftarrow V \cup \{\text{head}(r)\}$; $C \leftarrow C \cup \{\text{head}(r)\}$; $W \leftarrow W \setminus \{\text{head}(r)\}$
15: **else if** head$(r) \in C$ **then** $V \leftarrow V \cup \{\text{head}(r)\}$
16: **else** $Y \leftarrow Y \cup \{\text{head}(r)\}$
▷ b) Backward chaining with waiting facts
17: **else if** $F \leftarrow$ getLast$(W) \neq$ null **then**
18: **if** F is explicit and $F \notin E_1^-$, or $F \in Y$ **then**
19: $V \leftarrow V \cup \{F\}$; $C \leftarrow C \cup \{F\}$; $W \leftarrow W \setminus \{F\}$
20: **else if** $\exists r \in \rho(P, I)$: body$(r) \cap D = \emptyset$ and body$(r) \setminus (W \cup C) \neq \emptyset$ and head$(r) = F$
21: **then** $W \leftarrow W \cup (\text{body}(r) \setminus (W \cup C))$
22: **else** $C \leftarrow C \cup \{F\}$; $W \leftarrow W \setminus \{F\}$
▷ c) Determine facts affected by deletion
23: **else if** $\exists r \in \rho(P, I)$: body$(r) \cap D \neq \emptyset$ and head$(r) \notin W \cup C$ **then**
24: $W \leftarrow W \cup \{\text{head}(r)\}$
▷ d) Delete facts that cannot be proven
25: **else if** $(C \setminus V) \not\subseteq D$ **then** $D \leftarrow D \cup (C \setminus V)$
▷ e) Finish deletions and prepare insertions
26: **else** $I \leftarrow (I \setminus D) \cup E_1^+$; phase $\leftarrow 2$
27: **if** phase = 2 **then** ▷ INSERTIONS
▷ f) Add new derivable facts
28: **if** $\exists r \in \rho(P, I)$: head$(r) \notin I$ **then**
29: **if** $\exists F \in$ body(r) : F is explicit and marked **then** mark({head(r)})
30: $I \leftarrow I \cup \{\text{head}(r)\}$
▷ g) Prepare next update processing
31: **else**
32: $C \leftarrow \emptyset$; $D \leftarrow E_2^-$; $V \leftarrow \emptyset$; $W \leftarrow$ getMarked$(I) \setminus E_2^-$; $Y \leftarrow \emptyset$; unmark(I)
33: write I to output; $U_1 \leftarrow U_2$; $U_2 \leftarrow$ null; phase $\leftarrow 1$
34: **until** $\hat{U}$ ends and U_1 = null

For the next update U_2, however, we only look once per iteration, before we continue with the processing of U_1 to avoid any stagnation (cf. line 8). When a next update has been found, we directly use it to mark explicit facts that are in the dataset or added by the current update but also deleted by the next update (see line 9). To ensure a fast integration of the next update once it is available, we apply

at most one rule in each iteration as indicated by the existential quantifiers in lines 11, 20, 23, and 28. Note that despite the usage of the set $\rho(P, I)$ in those lines, we do not have to determine every possible rule instance at once, but instead can compute them gradually as needed.

The remaining operations in the algorithm are separated into a deletion and an insertion phase based on a phase variable (cf. lines 10 and 27), which is updated once the computations for a phase are finished (cf. lines 26 and 33). Similarly to the original B/F algorithm, we use various sets in the deletion phase to indicate that a fact is "completely checked" (C), "deleted" (D), "proven" (V), "waiting to be completely checked" (W), or "delayed" (Y), i.e., proven but not checked yet. The deletion phase can be further divided into five blocks (a–e) and the insertion phase into two (f–g), as indicated by the comments on the right-hand side of Algorithm 1. Note that the order of the blocks in the algorithm is based on their priority and does not necessarily represent their appearance during processing. We use the following example as starting point to describe the functionality of the different sets and blocks:

Example 3. *Assume we have four rules $p_1(x), p_2(x) \rightarrow q(x)$, $p_3(x) \rightarrow q(x)$, $q(x) \rightarrow r(x)$, and $q(x), p_4(x) \rightarrow s(x)$, and two consecutive updates $U_a = (\{p_1(c)\}, \{p_4(c)\})$ and $U_b = (\{p_4(c)\}, \emptyset)$, which we sequentially apply on a materialized dataset $I = \{p_1(c), p_2(c), p_3(c), q(c), r(c)\}$. During the first loop iteration in Algorithm 1, we assign U_a to U_1 and set $D = \{p_1(c)\}$, before we assign U_b to U_2 and then mark the fact $p_4(c)$.*

Since the sets V and W are initially empty, but D is not, we begin with block c), where we *determine facts that are affected by a deletion*. For that, we search for rule instances where the body contains some deleted fact from D and the head has not been checked yet (see line 23), in which case the head is added to the set of waiting facts W (see line 24). In our example, we use the rule instance $p_1(c), p_2(c) \rightarrow q(c)$ with $p_1(c) \in D$ to add $q(c)$ to W.

Due to $q(c) \in W$, we apply block b) in the next iteration, where we process facts that are waiting to be checked for a deletion-free derivation. For the implicit fact $q(c)$, we do this by applying *backward chaining* based on the rule instance $p_3(c) \rightarrow q(c)$, where the body does not contain any deleted fact from D but still consists of some facts that were not checked yet, so that they may be added to W for future processing (see lines 20 and 21). In particular, we only add facts to W if they do not already occur in $W \cup C$ to guarantee termination for recursive derivation cycles. As in the original B/F algorithm, we perform backward chaining in a depth-first manner by always selecting the most recently added fact from W (see line 17). Accordingly, we continue in the next iteration with the new fact $p_3(c)$. Because $p_3(c)$ is explicit and not deleted, it is regarded as "proven" and added to the set V. Thus, we do not need to process it any further and move it from W to the set of completed facts C (cf. lines 18 and 19).

With $p_3(c) \in V$, the next iteration starts with block a), which is responsible for the *forward chaining* segment of B/F, where we derive facts entirely with proven facts from V (cf. line 11). In addition, the derived fact is marked if the body of the applied rule contains a marked explicit fact (see line 12). Using the rule $p_3(x) \rightarrow q(x)$, we derive and, thus, prove the fact $q(c)$, which is also added to V as it has already been checked during backward chaining and, hence, occurs in W (or C), based on lines 13 to 15. Prioritizing forward over backward chaining facilitates a quick proving of facts which were considered during backward chaining, so that we can prevent their further processing in block b) by removing them from W (see line 14). With $q(c)$ in V, we then also derive $r(c)$ based on the rule $q(x) \rightarrow r(x)$ in the next iteration, but since $r(c)$ does not occur in W or C, it is added to the set of delayed facts Y (see line 16), so that it may directly be proven should it later appear during backward chaining (cf. lines 18 and 19).

At this point in our example, we computed every derivable fact from V, the set W is empty, we considered every rule instance that contains a fact from D, and every checked fact has been proven, i.e., $C = V$. Therefore, we can only apply block e), where we remove the facts in D from the materialized dataset I and add the new explicit facts E_1^+ of the current update U_1, resulting in $I = \{p_2(c), p_3(c), q(c), r(c), p_4(c)\}$, before we adapt the phase variable to continue with the insertion phase (see line 26).

In the first block f) of the insertion phase, we repeatedly add every fact to the dataset that is not present yet, but occurs as head in a rule instance (see lines 28 and 30). In our case, we use $q(c), p_4(c) \rightarrow s(c)$ to

add $s(c)$ to the dataset. Furthermore, we also mark the derived fact $s(c)$ due to the marked explicit fact $p_4(c)$ in the rule body (cf. line 29). Once we cannot add any new fact, we end the insertion phase in block g) by writing the now fully updated materialized dataset to the output stream and prepare the processing of the next update in lines 32 and 33: we empty the sets C, V, and Y to reset the proven facts, initialize D with the explicitly deleted facts of the next update, and use the set of marked implicit facts as starting point for W. In our example, we get $I = \{p_2(c), p_3(c), q(c), r(c), p_4(c), s(c)\}$ with $D = \{p_4(c)\}$ and $W = \{s(c)\}$.

Unlike the processing of the first update, the second one now begins with block b) as W is not empty. More specifically, we can directly prevent rule applications in block c), where we only consider a rule instance if the head does not already occur in W or C (see line 23). Accordingly, the initialization of W based on marked facts is how we improve the performance of B/F in the end. Since $s(c) \in W$ is neither explicit nor in Y, and there is no rule instance without any body fact in D which has $s(c)$ as head, the fact is moved to C based on line 22.

Because V and W are empty, and $s(c)$ occurs in C but not in V, we apply block d) next. At this point, we know that facts which were checked but could not be proven do not possess any alternative derivation and, hence, have to be deleted, which is why we add them to D, so that we may determine further facts affected by deletions in the following iterations (cf. line 25). Since $s(c)$ cannot be matched to any body atom in our rules, we cannot perform block c) and instead move on to block e), where the deleted facts are removed from the dataset, leading to $I = \{p_2(c), p_3(c), q(c), r(c)\}$. As U_b does not insert new facts, we then finish with block g).

5. Evaluation

Extending DRed with marking allowed us to reduce the CPU time that is needed to process a stream of updates by about 25% in average in the conducted evaluation [6]. For the B/F algorithm, we expect less improvement in the performance. The first reason is that marks are only computed based on explicit facts, which might lead to fewer marked implicit facts and, thus, a smaller reduction of rule applications. In addition, the main effort of B/F usually lies in the backward and forward chaining of the deletion phase, whereas determining what facts are affected by a deletion is comparatively simple. To see if the marking approach can still improve B/F despite those limitations, we conducted the same evaluation as for DRed with Marking.

5.1. Test Data

The evaluation involves two synthetic and one real test case. In the synthetic ones, we work with a graph represented by a set of directed edges, where randomly generated updates appropriately add and delete the same number of edges. In one case, we use a Datalog program P_{trans}, which computes (transitive) paths in the graph, and in the other one a program P_{seq}, where predicates are repeatedly renamed to create simple sequences of rule applications. In particular, P_{trans} allows for many alternative derivations, which is often an advantage for B/F compared to DRed, while facts in P_{seq} can only be derived in one specific way, which is usually less optimal for B/F.

The real test case is inspired by a task in autonomous driving, where we reason about dynamically loaded map data. Given a GPS track, we load map data within a radius of $50\,m$ around each GPS position and then generate a stream of updates, which states how the map data changes between the sequential GPS positions. The map data focuses on specific types of ways, for which Datalog rules are utilized to compute connections and their transitive relations. The exact rules of each program are listed in Appendix B.

Table 1
Test measurements for the synthetic data with P_{trans}

	time [in seconds]		reduction [in %]	deletions		backward		forward		inser-tions	marks	
size	old	new		old	new	old	new	old	new	both	ex.	im.
10	42.48	39.17	7.8	1,603	447	11,253	14,447	18,777	19,001	545	480	991
20	38.55	35.68	7.4	3,046	594	10,468	13,199	15,373	15,516	2,912	960	2,277
30	38.64	35.57	7.9	5,146	1,100	10,206	11,856	11,154	11,224	7,288	1,399	4,153
40	41.35	40.17	2.9	4,755	682	10,834	13,036	12,560	12,612	6,575	1,816	4,069
50	43.60	45.09	-3.4	3,778	360	11,514	14,441	15,060	15,096	4,570	2,215	3,660
60	45.78	48.51	-6.0	3,227	213	11,842	15,254	16,518	16,547	3,327	2,604	3,428
70	48.34	52.76	-9.1	2,724	161	11,919	16,054	17,780	17,796	2,158	2,973	3,085
80	50.04	55.76	-11.4	2,584	126	11,863	16,323	18,169	18,183	1,782	3,293	3,078

5.2. Test Execution and Results

We performed the evaluation by extending the implementation of DRed with Marking, which is available online[1]. The tested update streams are created with Java, while Constraint Handling Rules [15] based on SWI-Prolog[2] are used to implement both the original and our extended B/F algorithm. For a more efficient implementation, we slightly adapted Algorithm 1 to allow more than one rule application in a loop iteration once the next update U_2 has been provided.

During the tests, we measured the time spent by the CPU to process the whole stream, along with the number of applied rules in the different phases of the algorithm. For our extended B/F approach, we additionally counted the number of marked facts. The following results were obtained on a Windows 11 PC with an AMD Ryzen 7 3700X 3.59 GHz CPU and 16 GB RAM, using SWI-Prolog 9.3.15 with a 4 GB stack.

5.2.1. Synthetic Data Results

For the synthetic tests, we computed average values from three runs based on different random seeds and five repetitions. Every stream included exactly 50 updates, where the first one added 100 facts to an empty dataset, for which the materialization was first completed before any other update was considered. Each test run processed eight update streams, where the update size, i.e., the number of both added and deleted facts, increased evenly from 10 to 80. The maximum number of nodes in the graph was set to 20 for P_{trans} and to 100 for P_{seq}.

Table 1 shows the results for P_{trans}, where "old" refers to classical B/F and "new" to B/F with marking, respectively. Furthermore, "deletions" presents the number of rule applications for line 23 in Algorithm 1, "backward" for line 20, "forward" for line 11, and "insertions" for line 28. As indicated by the "reduction" column, the marking approach can both decrease and increase the processing time. Nevertheless, this may still be seen as an improvement, because B/F is generally suited for smaller updates [8], all of which benefit in our test.

The reason for the performance gain is provided by the "deletions" column, which shows a greatly reduced number of rule applications. For larger updates, however, this does not suffice as more facts in an update also mean more marking-related overhead. In addition, the ratio of marked implicit facts ("im.") to marked explicit facts ("ex.") is lower than for smaller updates, while the number of performed deletion rules in the original B/F is also relatively small for larger updates, thus further hindering the gain of the marking approach. Moreover, the number of applied rules during backward and forward

[1] https://github.com/M-Illich/dred-mark-eval
[2] https://www.swi-prolog.org/

Table 2
Test measurements for the synthetic data with P_{seq}

	time [in seconds]		reduction [in %]	deletions		backward	forward	insertions	marks	
size	old	new		old	new	both	both	both	ex.	im.
10	10.32	8.73	15.4	1,960	1,480	0	0	2,360	480	480
20	18.69	15.68	16.1	3,920	2,960	0	0	4,317	960	960
30	28.59	23.33	18.4	5,880	4,440	0	0	6,265	1,440	1,440
40	43.14	34.60	19.8	7,827	5,910	0	0	8,228	1,917	1,917
50	64.62	52.67	18.5	9,709	7,332	0	0	10,173	2,377	2,377
60	107.12	85.82	19.9	11,588	8,744	0	0	12,129	2,844	2,844
70	152.29	120.46	20.9	13,457	10,147	0	0	14,069	3,311	3,311
80	220.44	177.29	19.6	15,283	11,516	0	0	16,005	3,767	3,767

Table 3
Test measurements for the real data

	time [in seconds]		reduction [in %]	deletions		backward		forward		insertions	marks	
	old	new		old	new	old	new	old	new	both	ex.	im.
$track_0$	114.58	106.22	7.3	859	754	4,329	4,400	3,299	3,299	4,638	910	70
$track_1$	20.57	20.29	1.4	684	544	2,697	2,733	2,317	2,317	2,653	1,208	119
$track_2$	10.46	10.24	2.2	558	400	1,484	1,479	1,206	1,206	1,704	1,017	137

chaining is actually higher with marking, especially for larger updates, which may be explained by the change of order in which facts are checked for alternative derivations in line 17 of Algorithm 1, due to the initialization of the set W based on marked implicit facts in line 32. The impact of the facts' processing order on the number of performed rules is also the reason why the number of marked implicit facts does generally not match with the number of prevented deletion rules. For example, a fact might be already proven during the processing of another fact, before we determine that it is affected by a deletion.

The results for P_{seq} are in Table 2. Due to the lack of alternative derivations, we cannot apply any rules during the backward and forward chaining parts, which is why the prevention of rule applications indicated in the "deletions" column has a higher influence on the performance improvement than for P_{trans} (see "reduction" column), although P_{trans} could decrease the number of deletion rules much more. In particular, even the larger updates did benefit from the marking approach. For the same reason, the number of marked implicit facts equals the number of prevented deletion rules now.

5.2.2. Real Data Results

For the real test, we used predefined GPS tracks ($track_0$, $track_1$, $track_2$) to generate three streams consisting of 55, 83, and 114 updates that add/delete around 18/17, 14/14, and 9/8 facts on average, and 172/114, 152/167, and 65/74 facts at maximum, respectively. The results in Table 3 lead to similar conclusions as the synthetic tests, with an overall better performance for the marking approach (see "reduction" column) due to a decreased number of rule applications related to "deletions". While the undesired increase of applied backward chaining rules is much smaller than for P_{trans}, so is the number of marked implicit facts, especially in proportion to the explicit ones, which is why we mainly obtain small time improvements.

6. Conclusion

We extended the Backward/Forward algorithm with marking as done previously for Delete/Rederive. During the processing of an update, we directly take a look at the next available one and mark facts that are deleted by this next update. With those marked explicit facts, we can already determine some facts that have to be checked for alternative derivations during the processing of the next update, without the need to apply rules for that. An evaluation based on both synthetic and real data tests showed that the marking approach can accelerate the processing time, although improvements are often small. Future work may involve further optimizations, like integrating heuristics to specify the processing order of facts to prevent additional rule applications during backward and forward chaining, for instance.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

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

[2] B. Motik, Y. Nenov, R. Piro, I. Horrocks, D. Olteanu, Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 28, AAAI Press, 2014, pp. 129–137. URL: https://doi.org/10.1609/aaai.v28i1.8730.

[3] D. Carral, L. González, P. Koopmann, From Horn-SRIQ to Datalog: A Data-Independent Transformation That Preserves Assertion Entailment, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, AAAI Press, 2019, pp. 2736–2743. URL: https://doi.org/10.1609/aaai.v33i01.33012736.

[4] A. Gupta, I. S. Mumick, V. S. Subrahmanian, Maintaining Views Incrementally, in: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, ACM Press, 1993, pp. 157–166. URL: https://doi.org/10.1145/170035.170066.

[5] B. Motik, Y. Nenov, R. E. F. Piro, I. Horrocks, Incremental Update of Datalog Materialisation: the Backward/Forward Algorithm, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 29, AAAI Press, 2015, pp. 1560–1568. URL: https://doi.org/10.1609/aaai.v29i1.9409.

[6] M. Illich, B. Glimm, Delete/Rederive with Marking for Update Streams, in: The Semantic Web, volume 15718 of *Lecture Notes in Computer Science*, Springer Nature Switzerland, Cham, 2025, pp. 152–168. URL: https://doi.org/10.1007/978-3-031-94575-5_9.

[7] F. Bry, N. Eisinger, T. Eiter, T. Furche, G. Gottlob, C. Ley, B. Linse, R. Pichler, F. Wei, Foundations of Rule-Based Query Answering, Springer Berlin Heidelberg, 2007, pp. 1–153. doi:10.1007/978-3-540-74615-7_1.

[8] B. Motik, Y. Nenov, R. Piro, I. Horrocks, Maintenance of Datalog Materialisations Revisited, Artificial Intelligence 269 (2019) 76–136. URL: https://doi.org/10.1016/j.artint.2018.12.004.

[9] P. Hu, B. Motik, I. Horrocks, Optimised Maintenance of Datalog Materialisations, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, AAAI Press, 2018, pp. 1871–1879. URL: https://doi.org/10.1609/aaai.v32i1.11554.

[10] M. Terdjimi, L. Médini, M. Mrissa, Web Reasoning Using Fact Tagging, in: Companion Proceedings of the The Web Conference 2018, WWW '18, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2018, p. 1587–1594. doi:10.1145/3184558.3191615.

[11] Y. Ren, J. Z. Pan, Optimising Ontology Stream Reasoning with Truth Maintenance System, in: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, ACM Press, 2011, pp. 831–836. URL: https://doi.org/10.1145/2063576.2063696.

[12] J. Urbani, A. Margara, C. J. H. Jacobs, F. van Harmelen, H. E. Bal, DynamiTE: Parallel Materialization

of Dynamic RDF Data, in: The Semantic Web - ISWC 2013, volume 8218 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 657–672. URL: https://doi.org/10.1007/978-3-642-41335-3_41.

[13] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, M. Grossniklaus, Incremental Reasoning on Streams and Rich Background Knowledge, in: The Semantic Web: Research and Applications, volume 6088 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 1–15. URL: https://doi.org/10.1007/978-3-642-13486-9_1.

[14] D. Dell'Aglio, E. D. Valle, Incremental Reasoning on RDF Streams, in: Linked Data Management, Chapman and Hall/CRC, 2014, pp. 413–435. URL: http://dellaglio.org/preprints/14b1.pdf.

[15] T. Frühwirth, Constraint Handling Rules, Cambridge University Press, 2009.

A. Algorithm – Correctness

We show correctness of Algorithm 1 in a similar way to DRed with Marking. First, we prove that the algorithm works correctly if the stream only consists of a single update, and then show that the marking-related computations do not have any influence on both the current update's and the next update's dataset. For the following, we define that given two sequential updates U_a and U_b, the "processing of U_a" refers to loop iterations in Algorithm 1 where $U_1 = U_a$ and $U_2 = U_b$, while the "processing of U_b" refers to the subsequent, continuing iterations where $U_1 = U_b$. Furthermore, we say that a "derivation path" for a fact F is a sequence of rule instances $r_1, ..., r_n$, where $\mathsf{body}(r_1)$ only contains explicit facts, $\mathsf{head}(r_n) = F$, and $\mathsf{head}(r_i) \in \mathsf{body}(r_{i+1})$ for $1 \leq i < n$.

Let $I = \mathsf{mat}(P, E)$ be the initial dataset and $I' = \mathsf{mat}(P, E')$ be the correctly adapted dataset with $E' = (E \setminus E^-) \cup E^+$ for an update $U = (E^-, E^+)$. We prove the correct processing of a single update by showing that the algorithm's deletion phase removes every fact from $I \setminus I'$, while the insertion phase adds every new derivable fact from $I' \setminus I$, respectively. For the deletion phase, we first show the following claims related to Algorithm 1:

Claim 1. *A fact F is added to the set W if and only if it is also added to the set C.*

Proof. ($\Rightarrow$) When a fact F occurs in W, it will eventually be selected in line 17. If F satisfies the conditions in line 18, it will be added to C in the next line. The second condition, in line 20, can only be satisfied for a limited number of times due to the finite number of rule instances and the prevention of repetitions as the consequence of this operation contradicts its condition. Hence, F will eventually be added to C based on line 22. The only way to prevent the selection of F in line 17 is its removal from W. Whenever we remove a fact from W in lines 14, 19, and 22, however, we also add the fact to C.

($\Leftarrow$) We only add a fact F to C in lines 14, 19, and 22, which can only be visited when F is also in W (see lines 13 and 17). □

Claim 2. *A fact F is added to the set V if and only if $F \in C \cap I'$.*

Proof. ($\Rightarrow$) A fact F is only added to V in lines 14 and 19, where F is also added to C, or in line 15, which already requires that $F \in C$. The conditions for the former additions are that F is explicit and not in E^-, or that F can be derived by proven facts from V. We show that $F \in V \Rightarrow F \in I'$ by induction:

As base case, we consider the initial situation where V is empty, so that we can only add explicit facts that are not in E^- and, thus, still in $E' \subseteq I'$. As induction step, we consider the case where we add F by deriving it from facts in V. By our hypothesis, every fact from V is also in I', so that the facts derived from V have to be in I' too.

($\Leftarrow$) If a fact F occurs in $C \cap I'$, it was also processed as part of W based on Claim 1. If F is explicit, it cannot be in E^- due to $F \in I'$, so that it will be added to V in line 19. We show that an implicit F is added to V by induction over the length of F's derivation paths in $I \cap I'$, because implicit facts in C are based on rule instances from $\rho(P, I)$:

For the base case, we assume that F has a derivation path of length 1, which means that there exists a rule instance r with $\mathsf{head}(r) = F$ and $\mathsf{body}(r) \subseteq E \cap E'$. Since F was processed as part of W and is not explicit, we can apply r in line 20 to add every body fact to W. These body facts are in $E \cap E'$, so that they will be added to V in line 19 during the following algorithm iterations. Hence, we can also apply r in line 11 and, thus, add F to V as well.

For the induction step, we assume that F has a (shortest) derivation path of length $n+1$. Accordingly, there exists a rule instance r with $\mathsf{head}(r) = F$, where every fact in $\mathsf{body}(r)$ has a (shortest) derivation path of length $\leq n$. Due to $F \in I'$, we also have $\mathsf{body}(r) \subseteq I'$. By Claim 1, F occurred in W, in which case we can apply r in line 20 to add $\mathsf{body}(r)$ to W and, hence, to C. Thus, we have $\mathsf{body}(r) \subseteq C \cap I'$ and consequently $\mathsf{body}(r) \subseteq V$ based on our hypothesis. This allows us to apply r in line 11 and, thus, add F to V. □

With the above claims, we can show the correct behavior of the deletion phase:

Claim 3. *A fact F is added to the set D if and only if $F \in I \setminus I'$.*

Proof. ($\Rightarrow$) If $F \in D$ and F is explicit, then $F \in E^-$ (cf. line 6) and, thus, $F \in I \setminus I'$. For the case that F is implicit, we know that $F \in C$ and $F \notin V$ based on line 25. By Claim 2, this also requires that $F \notin I'$. Since the algorithm only works with rule instances from $\rho(P, I)$, we also have $C \subseteq I$, and hence $F \in I \setminus I'$.

($\Leftarrow$) If $F \in I \setminus I'$ and F is explicit, it has to appear in E^- and, thus, in D based on line 6. For the case that F is implicit, we use a proof by induction over the derivation path length of F:

For the base case, we assume that F has a derivation path of length 1, which means that there is a rule instance r, where $\mathsf{head}(r) = F$ and every fact from $\mathsf{body}(r)$ is explicit. Since $F \in I \setminus I'$, the body has to contain a fact from E^-, otherwise F would still be derivable in I'. Initially, we have $D = E^-$, so that we can apply r in line 23 and add F to W in the next line. By Claim 1, F will be added to C too. Because $F \notin I'$, we get $F \notin V$ by Claim 2, so that F is added to D based on line 25.

For the induction step, we assume that F has a (shortest) derivation path of length $n+1$. Accordingly, there exists a rule instance r with $\mathsf{head}(r) = F$, where every fact in $\mathsf{body}(r)$ has a (shortest) derivation path of length $\leq n$. Due to $F \in I \setminus I'$, we have $\mathsf{body}(r) \cap (I \setminus I') \neq \emptyset$. By our hypothesis, we then get $\mathsf{body}(r) \cap D \neq \emptyset$, so that we can apply r in line 23 to add F to W and, consequently, to C by Claim 1. Due to $F \notin I'$, we get $F \notin V$ by Claim 2, so that F will be added to D based on line 25. □

Next, we show the correct behavior of the algorithm's insertion phase:

Claim 4. *Let I^* be the updated dataset I of line 26, and E^* the corresponding set of explicit facts. When we reach line 33, we have $I = \mathsf{mat}(P, E^*)$.*

Proof. As long as $\mathsf{phase} = 2$, we repeatedly look for rule instances in I where the head is not yet present, and add this head to I (cf. lines 28 and 30). Accordingly, we extend I with $P(I)$, as defined in Section 2. Repeating this until the dataset does not change anymore, i.e., once we cannot add any new fact, will eventually lead to $I = \mathsf{mat}(P, E^*)$ as defined in Section 2.1. □

Based on this, we can now prove that the algorithm correctly processes a single update:

Lemma 1 (Single update)**.** *Given a Datalog program P, a materialized dataset $I = \mathsf{mat}(P, E_0)$, and a stream of updates $\hat{U} = \langle U \rangle$ with $U = (E^-, E^+)$, Algorithm 1 returns as output a stream $\langle \mathsf{mat}(P, E_1) \rangle$ with $E_1 = (E_0 \setminus E^-) \cup E^+$.*

Proof. By Claim 3, we obtain $D = \mathsf{mat}(P, E_0) \setminus \mathsf{mat}(P, E_1)$ once we cannot extend D any further, in which case line 26 is executed and every fact that cannot be derived anymore in $\mathsf{mat}(P, E_1)$ is deleted. In the same line, we also add E^+ to I, so that we eventually obtain $I = \mathsf{mat}(P, E_1)$ once we reach line 33, based on Claim 4. □

Knowing that Algorithm 1 works correctly in the classical way, where we do not take a look at the next update, we next show that the marking-related computations do not interfere with the computation of the dataset. First, we consider the case for the current update U_1:

Lemma 2 (No influence on current dataset)**.** *Given two sequential updates U_a and U_b from $\hat{U}$ in Algorithm 1, the marks that are computed due to U_b during the processing of U_a do not influence the dataset I that is returned at the end of processing U_a.*

Proof. The procedures MARK, UNMARK and the function GETMARKED do neither add nor remove facts. The only cases where a marked fact is required as a condition to perform some operation are in lines 12 and 29, where we just mark a fact and, thus, do not change the dataset. □

To prove that the dataset of the next update U_2 will not be affected by the markings either, we first show the following claim:

Claim 5. *The order in which we add facts to the set W does not have an influence on the dataset I that is returned at the end of an update's processing.*

Proof. Lines 20 and 23 do not specify what rule we may choose if several ones are applicable, so that the order in which we add facts to W can be different each time we use the algorithm. Based on Lemma 1, however, we know that the algorithm still produces the correct materialized dataset. □

Now, we can prove the following lemma:

Lemma 3 (No influence on next dataset)**.** *Given two sequential updates U_a and U_b from $\hat{U}$ in Algorithm 1, the marks that are computed due to U_b during the processing of U_a do not influence the dataset I that is returned at the end of processing U_b.*

Proof. Marked facts influence the next update's processing based on the initialization of the set W in line 32. Here, we only add implicit facts (by removing E_2^-), which can only be marked in lines 12 and 29. In both cases, we mark the head of a rule instance, where the body contains a marked explicit fact. Based on line 9, we only mark explicit facts that occur in E_2^-. Since we use E_2^- to initialize the set D when we prepare the next update's processing in line 32, we could apply those rule instances in line 23 during the next update's processing if we would start with an empty set W. This means that every fact that we add to W in line 32 based on markings, would also be added to W if we processed the next update in the classical way, where we consider each update alone, without any marking. By Claim 5, changing the order of fact additions to W based on its initialization does not affect the resulting dataset, so that we obtain the same dataset I at the end as if we would process the next update alone, which is correct based on Lemma 1. □

Combining the above lemmas, we can show correctness of Algorithm 1 as follows:

Theorem 1. *Given a Datalog program P, a materialized dataset $I = \mathsf{mat}(P, E_0)$, and a stream of updates $\hat{U} = \langle U_1, U_2, ... \rangle$, Algorithm 1 returns as output a stream $\langle \mathsf{mat}(P, E_1), \mathsf{mat}(P, E_2), ... \rangle$ with $E_i = (E_{i-1} \setminus E_i^-) \cup E_i^+$ for each $U_i = (E_i^-, E_i^+) \in \hat{U}$.*

Proof. We prove this by induction: As base case, we have the first update $U_1 \in \hat{U}$, for which the algorithm correctly computes $\mathsf{mat}(P, E_1)$ based on Lemmas 1 and 2. In the induction step, we consider two sequential updates $U_i, U_{i+1} \in \hat{U}$. By hypothesis, we get the correct result for U_i. By Lemma 3, the related computations do not affect the result of U_{i+1}, so that we also obtain the correct dataset $\mathsf{mat}(P, E_{i+1})$ by hypothesis. □

B. Evaluation – Datalog Programs

$$
\begin{aligned}
\mathit{edge}(x, y) &\rightarrow \mathit{path}(x, y) \\
\mathit{edge}(x, y), \mathit{path}(y, z) &\rightarrow \mathit{path}(x, z)
\end{aligned}
$$

(a) Rules for Datalog program P_{trans}

$$
\begin{aligned}
\mathit{edge}(x, y) &\rightarrow \mathit{edge1}(x, y) \\
\mathit{edge1}(x, y) &\rightarrow \mathit{edge2}(x, y) \\
\mathit{edge2}(x, y) &\rightarrow \mathit{edge3}(x, y) \\
\mathit{edge3}(x, y) &\rightarrow \mathit{edge4}(x, y)
\end{aligned}
$$

(b) Rules for Datalog program P_{seq}

Figure 1: Rules for the Datalog programs used in synthetic data tests

$$
\begin{aligned}
\mathit{nextInWay}(x, y_1, z_1), \mathit{nextInWay}(x, y_2, z_2), neq(z_1, z_2) &\rightarrow \mathit{connection}(z_1, z_2) \\
\mathit{nextInWay}(x, y_1, z_1), \mathit{nextInWay}(x_2, x, z_2), neq(z_1, z_2) &\rightarrow \mathit{connection}(z_1, z_2) \\
\mathit{nextInWay}(x_1, y, z_1), \mathit{nextInWay}(y, y_2, z_2), neq(z_1, z_2) &\rightarrow \mathit{connection}(z_1, z_2) \\
\mathit{nextInWay}(x_1, y, z_1), \mathit{nextInWay}(x_2, y, z_2), neq(z_1, z_2) &\rightarrow \mathit{connection}(z_1, z_2) \\
\mathit{connection}(x, y), \mathit{connection}(y, z) &\rightarrow \mathit{connection}(x, z)
\end{aligned}
$$

with $neq(x, y) := x \neq y$

Figure 2: Rules for the Datalog program used in real data tests

The atom $\mathit{nextInWay}(x, y, z)$ states that the node x is followed by the node y on the way z.