Incremental Update of Datalog Materialisation: the Backward/Forward Algorithm

Boris Motik, Yavor Nenov, Robert Piro and Ian Horrocks
Department of Computer Science, Oxford University
Oxford, United Kingdom
firstname.lastname@cs.ox.ac.uk

Abstract

Datalog-based systems often *materialise* all consequences of a datalog program and the data, allowing users' queries to be evaluated directly in the materialisation. This process, however, can be computationally intensive, so most systems update the materialisation incrementally when input data changes. We argue that existing solutions, such as the well-known *Delete/Rederive* (DRed) algorithm, can be inefficient in cases when facts have many alternate derivations. As a possible remedy, we propose a novel *Backward/Forward* (B/F) algorithm that tries to reduce the amount of work by a combination of backward and forward chaining. In our evaluation, the B/F algorithm was several orders of magnitude more efficient than the DRed algorithm on some inputs, and it was never significantly less efficient.

1 Introduction

Datalog (Abiteboul, Hull, and Vianu 1995) is a widely used rule language capable of expressing recursive dependencies. A prominent application of datalog is answering queries over ontology-enriched data: datalog can capture OWL 2 RL (Motik et al. 2012) ontologies extended with SWRL (Horrocks et al. 2004) rules, so modern Semantic Web systems often use datalog 'behind the scenes'.

When the performance of query answering is critical, datalog systems often *materialise* (i.e., compute and explicitly store) all consequences of datalog rules and the data in a preprocessing step so that users' queries can then be evaluated directly over the stored facts; systems such as Owlgres (Stocker and Smith 2008), WebPIE (Urbani et al. 2012), Oracle's RDF store (Wu et al. 2008), OWLIM SE (Bishop et al. 2011), and RDFox (Motik et al. 2014) all use this technique. Materialisation is computationally intensive, so recomputing all consequences whenever input data changes is usually unacceptable, and most systems employ an *incremental maintenance* algorithm: given a datalog program Π, a set E of explicitly given facts, materialisation $I = \Pi^\infty(E)$ of E w.r.t. Π, and sets of facts E^- and E^+ to be deleted and added, respectively, such an algorithm uses the precomputed set I to compute $\Pi^\infty((E \setminus E^-) \cup E^+)$ more efficiently. Note that we consider only updates of explicitly given facts: changing derived facts does not have a unique semantics and is usually much harder. Fact insertion can be efficiently handled using the standard seminaïve algorithm (Abiteboul, Hull, and Vianu 1995): datalog (without negation-as-failure) is monotonic, so one can just 'continue' materialisation from E^+; hence, in this paper we usually assume that $E^+ = \emptyset$. In contrast, fact deletion is much more involved since one must identify and retract all facts in I not derivable from $E \setminus E^-$. Gupta and Mumick (1995) presents an extensive overview of the existing approaches to incremental update, which can be classified into two groups.

The approaches in the first group keep track of auxiliary information during materialisation to efficiently delete facts. In particular, *truth maintenance systems* (Doyle 1979; de Kleer 1986) track dependencies between facts; many can handle nonmonotonic datalog extensions, but their space and time overheads are often prohibitive on large datasets. Alternatively, one can count the derivations of facts during materialisation, and then delete facts when their derivation counter drops to zero (Gupta, Katiyar, and Mumick 1992); however, Nicolas and Yazdanian (1983) show in Section 6.2 that this technique may not correctly handle recursive rules. Urbani et al. (2013) applied this technique (with its inherent limitation regarding recursion) to the Semantic Web.

The approaches in the second group do not gather any additional information. The *Delete/Rederive* (DRed) algorithm (Gupta, Mumick, and Subrahmanian 1993; Staudt and Jarke 1996) generalises approaches by Harrison and Dietrich (1992), Küchenhoff (1991), and Urpí and Olivé (1992) and is the state of the art in this group. The DRed algorithm first overestimates and deletes all facts in I that might need to be deleted due to removing E^-, and then it rederives the facts still derivable from $E \setminus E^-$. The DRed algorithm has been adapted to RDF data (Barbieri et al. 2010; Urbani et al. 2013; Ognyanov and Velkov 2014) and incremental ontology classification (Ren and Pan 2011; Kazakov and Klinov 2013).

The overdeletion phase of the DRed algorithm can be a source of inefficiency. As a simple example, let Π contain rules (1)–(3), and let $E = \{A(a), B(a)\}$.

$$C_1(x) \leftarrow A(x) \tag{1}$$
$$C_1(x) \leftarrow B(x) \tag{2}$$
$$C_i(x) \leftarrow C_{i-1}(x) \quad \text{for } i \in \{2, \dots, n\} \tag{3}$$

Let $I = \Pi^\infty(E)$ be the materialisation of E w.r.t. Π, and let $E^- = \{A(a)\}$. Each fact $C_i(a) \in I$ is derived once

from $A(a)$ and once from $B(a)$; since the DRed algorithm overdeletes all consequences of E^-, it will delete all $C_i(a)$ only to later rederive them from $B(a)$. Overdeletion can thus be very inefficient when facts are derived more than once, and when facts contribute to many proofs of other facts. Such examples abound in the Semantic Web since, as we argue in more detail in Section 3, rules obtained from OWL ontologies often interact significantly with each other.

Thus, in Section 4 we present a novel *Backward/Forward* (B/F) algorithm. Our algorithm does not gather any information during materialisation and, instead of a potentially inefficient overdeletion phase, it determines whether deleted facts have alternate proofs from $E \setminus E^-$ using a combination of backward and forward chaining. On the example from the previous paragraph, B/F proves that $C_1(a)$ has an alternative derivation from $E \setminus E^- = \{B(a)\}$, and so it does not consider any of $C_i(a)$, $i > 1$. Thus, while the DRed algorithm explores the consequences of the facts in E^-, our algorithm explores their support, which is more efficient when proof trees are 'shallow'.

Our algorithm is applicable to general datalog programs. However, query answering over ontologies is a prominent application of datalog so, to evaluate our approach, we have implemented the DRed and the B/F algorithm in the open-source system RDFox (Motik et al. 2014), and have compared them on well-known Semantic Web benchmarks; we present our results in Section 5. In addition to the running times, we have also determined the numbers of facts and rule derivations of the two algorithms, which allowed us to compare the 'work' in an implementation-independent way. Our results show that the B/F algorithm typically outperforms the DRed algorithm, and that this difference is larger on datasets where a single fact has numerous derivations.

2 Preliminaries

A *term* is a *constant* or a *variable*. An *atom* has the form $R(t_1, \ldots, t_n)$, where R is an n-ary relation and each t_i is a term, for $0 \leq i \leq n$. A *fact* is an atom with no variables. A *(datalog) rule* r is an implication of the form

$$H \leftarrow B_1 \wedge \cdots \wedge B_n \tag{4}$$

where H is the *head* atom, $B_1, \ldots, B_n$ are *body* atoms, and each variable in H also occurs in some B_i. Let $\mathsf{h}(r) = H$, let $\mathsf{b}_i(r) = B_i$ for $1 \leq i \leq n$, and let $\mathsf{len}(r) = n$. A *(datalog) program* Π is a finite set of rules. A *substitution* σ is a partial mapping of variables to constants; for α a term, an atom, or a rule, $\alpha\sigma$ is the result of replacing each occurrence of x in α with $\sigma(x)$ when the latter is defined; if σ maps all variables in α, then $\alpha\sigma$ is an *instance* of α. For E, E^-, and E^+ finite sets of facts, the *materialisation* $\Pi^\infty(E)$ of E w.r.t. Π is the smallest superset of E such that $H\sigma \in \Pi^\infty(E)$ for each rule in Π of the form (4) and substitution σ with $B_i\sigma \in \Pi^\infty(E)$ for $1 \leq i \leq n$; moreover, $\Pi^\infty((E \setminus E^-) \cup E^+)$ is the *incremental update* of $\Pi^\infty(E)$ with E^- and E^+. A *derivation* is the process of matching the body atoms of a rule in a set of facts and extending it with the instantiated rule head.

In this paper, we do not distinguish intensional from extensional relations—that is, explicit facts can contain relations occurring in the rules' heads. This is a trivial modification of datalog: to assert a fact $R(\vec{a})$ where relation R occurs in the head of a rule, one can introduce a fresh relation R^{edb}, assert $R^{edb}(\vec{a})$, and add the rule $R(\vec{x}) \leftarrow R^{edb}(\vec{x})$. In contrast, we distinguish explicit from implicit facts and allow only the former to be deleted. Deleting implicit facts has been considered in the literature, but is much more complex and in fact there are many possible semantics of such updates. The DRed algorithm considers an identical setting.

3 Motivation

To motivate our work, we next recapitulate the DRed algorithm by Gupta, Mumick, and Subrahmanian (1993), discuss some of its drawbacks, and suggest possible improvements.

The Delete/Rederive Algorithm

We describe steps of the DRed algorithm using evaluation of datalog rules whose atoms refer to several sets of facts. Thus, for $R(\vec{x})$ an atom and X a set of facts, an *extended atom* has the form $R(\vec{x})^X$. If $R(\vec{x})^X$ occurs in a rule body, the atom is true under substitution σ if $R(\vec{x})\sigma \in X$; moreover, if $R(\vec{x})^X$ occurs in a rule head and the body of the rule is true under substitution σ, fact $R(\vec{x})\sigma$ derived by the rule is added to set X. This notation does not substantially extend datalog: we can simulate $R(\vec{x})^X$ by introducing a fresh relation R^X and writing $R^X(\vec{x})$ instead of $R(\vec{x})^X$; however, extended atoms offer a more convenient presentation style in Section 4.

Let Π be a program, let E, E^-, and E^+ be sets of facts, and let $I := \Pi^\infty(E)$. The DRed algorithm uses several auxiliary sets: set D accumulates facts that might need to be removed from I due to the removal of E^-, set A gathers facts derived by a single round of rule application, and set N contains facts derived in a previous round. The algorithm updates I to contain $\Pi^\infty((E \setminus E^-) \cup E^+)$ as follows.

DR1 Set $E := E \setminus E^-$, set $A := E^-$, and set $D := \emptyset$.

DR2 Set $N := A \setminus D$. If $N = \emptyset$, then go to step **DR3**; otherwise, set $D := D \cup N$, set $A := \emptyset$, for each rule r in Π of the form (4), evaluate rule (5) instantiated for each $1 \leq i \leq n$, and repeat step **DR2**.

$$H^A \leftarrow B_1^I \wedge \ldots \wedge B_{i-1}^I \wedge B_i^N \wedge B_{i+1}^I \wedge \ldots \wedge B_n^I \tag{5}$$

After this step, D contains all facts that might need to be deleted, so this step is called *overdeletion*.

DR3 Set $I := I \setminus D$, and set $A := (D \cap E) \cup E^+$.

DR4 For each rule in Π of the form (4), evaluate rule (6).

$$H^A \leftarrow H^D \wedge B_1^I \wedge \ldots \wedge B_n^I \tag{6}$$

This *rederivation* step adds to A facts that have an alternate derivation in the updated set I. Atom H^D in rule (6) restricts rederivation only to facts from step **DR2**.

DR5 Set $N := A \setminus I$. If $N = \emptyset$, then terminate; otherwise, set $I := I \cup N$, set $A := \emptyset$, for each rule in Π of the form (4), evaluate rule (5) instantiated for each $1 \leq i \leq n$, and repeat step **DR5**. This step is called *(re)insertion*.

Step **DR2** was presented by Gupta, Mumick, and Subrahmanian (1993) as materialisation of a program consisting of rules $H^D \leftarrow B_1^I \wedge \ldots \wedge B_{i-1}^I \wedge B_i^D \wedge B_{i+1}^I \wedge \ldots \wedge B_n^I$, and analogously for steps **DR4** and **DR5**. Our more explicit presentation style allows us to discuss certain issues.

Table 1: Running example

Program Π_{ex}

$TA(x) \leftarrow Person(x) \wedge Tutor(x,y) \wedge Course(y)$	(R1)
$Person(x) \leftarrow TA(x)$	(R2)
$Person(x) \leftarrow Tutor(x,y)$	(R3)
$Course(y) \leftarrow Tutor(x,y)$	(R4)

The set of explicitly given facts E

$Tutor(john, math)$	(E1)	$Tutor(john, phys)$	(E3)
$Tutor(peter, math)$	(E2)		

The derived facts from $I = \Pi_{ex}^{\infty}(E)$

$Person(peter)$	(I1)	$Person(john)$	(I4)
$TA(peter)$	(I2)	$TA(john)$	(I5)
$Course(math)$	(I3)	$Course(phys)$	(I6)

Problems with Overdeletion

As we have suggested in the introduction, the DRed algorithm can be inefficient on datasets where facts have more than one derivation and where they derive many other facts. This is often the case in programs obtained from OWL ontologies. To demonstrate the relevant issues, in Table 1 we present an example derived from the well-known University Ontology Benchmark (UOBM) (Ma et al. 2006).

Ontologies often define classes using axioms of the form (in description logic syntax) $A \equiv B \sqcap \exists R.C$. For example, axiom $TA \equiv Person \sqcap \exists Tutor.Course$ in UOBM says that teaching assistant (TA) is a person that teaches a course. OWL 2 RL systems translate such axioms into datalog by dropping existentially quantified parts; for example, Grosof et al. (2003) translate this axiom into rules (R1) and (R2). Moreover, UOBM also says that $Person$ and $Course$ are the domain and range, respectively, of $Tutor$, giving rise to rules (R3) and (R4), respectively. Rule (R1) could clearly be simplified to $TA(x) \leftarrow Tutor(x,y)$; however, doing this in practice is nontrivial because of interactions between many rules. One can readily check that materialising facts (E1)–(E3) w.r.t. Π_{ex} produces facts (I1)–(I6).

When the DRed algorithm is used to delete (E1), from (E1) step **DR2** derives (I4) by (R3), (I3) by (R4), and (I5) by (R1); then, from (I3) it derives (I2) by (R1); and from (I2) it derives (I1) by (R2). Thus, all implicit facts apart from (I6) get deleted in step **DR3**, only to be rederived in steps **DR4** and **DR5**. This problem is exacerbated by the fact that $Person$ occurs in many axioms of the form $A_i \equiv Person \sqcap \exists R_i.C_i$; hence, the effects of deleting (a fact that implies) a $Person$ fact often propagate through I.

Improving Overdeletion via Backward Chaining

In our algorithm, we improve step **DR2** by removing from N all facts with a proof from $E \setminus E^-$. In our example, fact (E1) does not have such a proof, so we add (E1) to N and D; then, step **DR2** derives (I3)–(I5). If we can now show that these facts have a proof from $E \setminus E^-$, no further overdeletion is needed; in fact, steps **DR3** and **DR4** become redundant since the deletion step is 'exact'. Hence, we can remove the unproved facts from I and process E^+ using step **DR5**.

Our B/F algorithm uses backward chaining to find a proof of, say, (I4) from $E \setminus E^-$. To this end, we try to match the heads of the rules in Π to (I4); for example, matching the head of (R3) produces a partially instantiated rule $r_1 = Person(john) \leftarrow Tutor(john, y)$. We then evaluate the body of r_1 against I to obtain *supporting* facts for (I4); fact (E3) is one such fact. We now recursively prove (E3), which is trivial since $E \setminus E^-$ contains (E3); but then, by r_1 we have a proof of (I4), as required. Note that the head of rule (R2) also matches (I4); however, since we already have a proof of (I4), we can stop the search.

This idea, however, may not terminate due to cyclic dependencies between the rules. For example, if we first match (I4) to the head of (R2), we will recursively try to prove (I5); but then, (I5) matches with the head of (R1) so we will recursively try to prove (I4), which creates a cycle. To address this problem, we apply backward chaining to each fact only once, collecting all such facts in a set C of *checked* facts. Backward chaining thus does not directly prove facts, but only determines the subset $C \subseteq I$ of facts whose status needs to be determined. We compute the set P of facts with a proof from $E \setminus E^-$ by forward chaining: we use rules (7) for each relation R in Π to obtain the directly proved facts $C \cap (E \setminus E^-)$, and we use rule (8) for each rule in Π of the form (4) to obtain the indirectly proved facts.

$$R(\vec{x})^P \leftarrow R(\vec{x})^C \wedge R(\vec{x})^E \tag{7}$$

$$H^P \leftarrow H^C \wedge B_1^P \wedge \ldots \wedge B_n^P \tag{8}$$

Such a combination of backward and forward chaining ensures termination, and we explore the candidate proofs for each fact only once, rather than all proof permutations.

The B/F algorithm thus essentially exchanges forward chaining over the *consequences* of E^- in step **DR2** for forward chaining over the *support* of E^- using rules (7)–(8). Our experiments in Section 5 show that the latter set of facts is often smaller than the former set; thus, since fewer facts give rise to fewer derivations, the B/F algorithm often outperforms the DRed algorithm in practice.

Repeating Derivations

Repeated derivations are a secondary source of inefficiency in the DRed algorithm. In step **DR5**, consider a rule $r \in \Pi$ of the form (4) and a substitution τ that matches two (not necessarily distinct) facts $F_i, F_{i'} \in N$ to body atoms B_i and $B_{i'}$, $i < i'$, of r—that is, $B_i\tau = F_i$ and $B_{i'}\tau = F_{i'}$; then, applying (5) for i and i' derives $H^A\tau$ twice. We can avoid this as in the seminaïve algorithm (Abiteboul, Hull, and Vianu 1995) by matching atoms $B_1, \ldots, B_{i-1}$ of rules (5) in $I \setminus N$, rather than in I; then, we derive $H^A\tau$ when evaluating rule (5) for i, but not for i' (since $F_i \in N$ so $F_i \notin I \setminus N$).

Step **DR2** is even more inefficient. Consider two sets of facts N_1 and N_2 where N_1 is processed before N_2, a rule $r \in \Pi$, and a substitution τ that matches two facts $F_i \in N_1$ and $F_{i'} \in N_2$ to body atoms B_i and $B_{i'}$, $i < i'$, of r; since $N_1 \subseteq I$ and $N_2 \subseteq I$, fact $H^A\tau$ is derived twice. Thus, each time we process some set N, we can repeat derivations

not only with facts in N (as in step **DR5**), but also in D. Note that this has no correspondence in the seminaïve algorithm, so we developed a novel 'inverse seminaïve' solution. In particular, we maintain a set of facts O that we extend with N after each iteration in step **DR2** (i.e., $O = D \setminus N$); moreover, we match atoms $B_1, \ldots, B_{i-1}$ of rules (5) in $I \setminus (O \cup N)$ and atoms $B_{i+1}, \ldots, B_n$ in $I \setminus O$. Thus, once we add a fact to O, we do not consider it again in (5).

4 The Backward/Forward Algorithm

We now formalise our ideas from Section 3. The Backward/Forward (B/F) algorithm takes as input a program Π, sets of facts E and E^-, and the materialisation $I = \Pi^\infty(E)$ of E w.r.t. Π, and it updates I to $\Pi^\infty(E \setminus E^-)$. We do not discuss fact addition since step **DR5** can handle it efficiently.

We use several auxiliary sets of facts. For X a set and F a fact, $X.\mathsf{add}(F)$ adds F to X, and $X.\mathsf{delete}(F)$ removes F from X; both operations return `true` if X was changed. To facilitate iteration over these sets, $X.\mathsf{next}$ returns a fresh fact from X, or ε if all facts have been returned.

To evaluate rule bodies in a set of facts X, we use a query answering mechanism that allows us to prevent repeated derivations. An *annotated query* is a conjunction $Q = B_1^{\bowtie_1} \wedge \ldots \wedge B_k^{\bowtie_k}$ where each B_i is an atom and *annotation* $\bowtie_i$ is either empty or equal to $\neq$. For X and Y sets of facts and σ a substitution, $X.\mathsf{eval}(Q, Y, \sigma)$ returns a set containing each smallest substitution τ such that $\sigma \subseteq \tau$ and, for $1 \leq i \leq k$, (i) $B_i\tau \in X$ if $\bowtie_i$ is empty or (ii) $B_i\tau \in X \setminus Y$ if $\bowtie_i$ is $\neq$; intuitively, τ extends σ so that each $B_i^{\bowtie_i}$ is matched in X or $X \setminus Y$, depending on the annotation $\bowtie_i$. Any join algorithm can implement this operation; for example, Motik et al. (2014) describe how to use index nested loop join. We often write $[Z \setminus W]$ instead of X, meaning that Q is evaluated in the difference of sets Z and W.

We must also be able to identify the rules in Π whose head atom or some body atom matches a given fact F. For the former, $\Pi.\mathsf{matchHead}(F)$ returns a set containing each tuple $\langle r, Q, \sigma \rangle$ where $r \in \Pi$ is a rule of the form (4), σ is a substitution such that $H\sigma = F$, and $Q = B_1 \wedge \ldots \wedge B_n$. For the latter, $\Pi.\mathsf{matchBody}(F)$ returns a set containing each tuple $\langle r, Q, \sigma \rangle$ where $r \in \Pi$ is a rule of the form (4), σ is a substitution such that $B_i\sigma = F$ for some $1 \leq i \leq n$, and Q is

$$Q = B_1^{\neq} \wedge \ldots \wedge B_{i-1}^{\neq} \wedge B_{i+1} \wedge \ldots \wedge B_n. \qquad (9)$$

For efficiency, the rules in Π should be indexed; for example, one can use the indexing scheme by Motik et al. (2014).

Intuitively, each $\langle r, Q, \sigma \rangle \in \Pi.\mathsf{matchHead}(F)$ identifies a match of F to the head of $r \in \Pi$; and for each substitution $\tau \in I.\mathsf{eval}(Q, \emptyset, \sigma)$, atoms $\mathsf{b}_i(r)\tau$ are candidates for deriving F from the updated set E via rule r. Analogously, each $\langle r, Q, \sigma \rangle \in \Pi.\mathsf{matchBody}(F)$ identifies a match of F to some body atom B_i in r, and each $\tau \in X.\mathsf{eval}(Q, \{F\}, \sigma)$ matches the body of r in X so that atoms $B_1, \ldots, B_{i-1}$ are not matched to F. The latter prevents repeated derivations: $\{F\}$ corresponds to the set N in steps **DR2** and **DR5**, so the annotation on $B_j^{\neq}$ for $1 \leq j < i$ effectively prevents B_j from being matched to $N = \{F\}$.

The B/F algorithm is given in Algorithm 1. We first update E and initialise set D (line 2). To identify deleted facts, in

Algorithm 1 $\mathsf{B/F{-}delete}()$

Global variables:
E: explicit facts — E^-: facts to delete from E
Π: a program — I: $\Pi^\infty(E)$
C: checked facts — D: examined consequences of E^-
Y: delayed facts — O: the processed subset of D
P: proved facts — V: the processed subset of P
S: disproved facts

1: $C := D := P := Y := O := S := V := \emptyset$
2: **for each** fact $F \in E^-$ **do** $E.\mathsf{delete}(F)$ and $D.\mathsf{add}(F)$
3: **while** $(F := D.\mathsf{next}) \neq \varepsilon$ **do**
4: $\quad \mathsf{checkProvability}(F)$
5: $\quad$ **for each** $G \in C \setminus P$ **do** $S.\mathsf{add}(G)$
6: $\quad$ **if** $F \notin P$ **then**
7: $\quad\quad$ **for each** $\langle r, Q, \sigma \rangle \in \Pi.\mathsf{matchBody}(F)$ **do**
8: $\quad\quad\quad$ **for each** $\tau \in [I \setminus O].\mathsf{eval}(Q, \{F\}, \sigma)$ **do**
9: $\quad\quad\quad\quad D.\mathsf{add}(\mathsf{h}(r)\tau)$
10: $\quad\quad O.\mathsf{add}(F)$
11: **for each** $F \in D \setminus P$ **do** $I.\mathsf{delete}(F)$

Algorithm 2 $\mathsf{checkProvability}(F)$

12: **if** $C.\mathsf{add}(F) = \mathsf{false}$ **then return**
13: $\mathsf{saturate}()$
14: **if** $F \in P$ **then return**
15: **for each** $\langle r, Q, \sigma \rangle \in \Pi.\mathsf{matchHead}(F)$ **do**
16: $\quad$ **for each** $\tau \in [I \setminus S].\mathsf{eval}(Q, \emptyset, \sigma)$ **do**
17: $\quad\quad$ **for each** i with $1 \leq i \leq \mathsf{len}(r)$ **do**
18: $\quad\quad\quad \mathsf{checkProvability}(\mathsf{b}_i(r)\tau)$
19: $\quad\quad$ **if** $F \in P$ **then return**

Algorithm 3 $\mathsf{saturate}()$

20: **while** $(F := C.\mathsf{next}) \neq \varepsilon$ **do**
21: $\quad$ **if** $F \in E$ or $F \in Y$ **then**
22: $\quad\quad P.\mathsf{add}(F)$
23: **while** $(F := P.\mathsf{next}) \neq \varepsilon$ **do**
24: $\quad$ **if** $V.\mathsf{add}(F)$ **then**
25: $\quad\quad$ **for each** $\langle r, Q, \sigma \rangle \in \Pi.\mathsf{matchBody}(F)$ **do**
26: $\quad\quad\quad$ **for each** $\tau \in V.\mathsf{eval}(Q, \{F\}, \sigma)$ **do**
27: $\quad\quad\quad\quad H := \mathsf{h}(r)\tau$
28: $\quad\quad\quad\quad$ **if** $H \in C$ **then** $P.\mathsf{add}(H)$
29: $\quad\quad\quad\quad$ **else** $Y.\mathsf{add}(H)$

lines 3–10 we repeatedly extract a fact F from D (line 3) and apply rules (5) (lines 7–9); this is similar to step **DR2** of the DRed algorithm, but with three important differences. First, we use Algorithm 2 to determine whether F has a proof from the updated set E; if so, we do *not* apply the rules (line 6). Second, our 'inverse seminaïve' strategy from Section 3 prevents repeated derivations: we exclude the set of processed facts O from the matches in line 8, and we add F to O in line 10. Third, we identify the set S of disproved facts (line 5) to optimise backward chaining. After we process all facts in D, all facts that need to be deleted are contained in D, and all facts of D that have a proof from E' are contained in P; hence, in line 11 we simply remove $D \setminus P$ from I.

Algorithm 2 implements backward chaining as we outlined in Section 3. In line 12 we ensure that each fact F is

checked only once, and in line 13 we update P to contain all facts in C that are provable from E using rules (7)–(8). Then, we identify each rule $r \in \Pi$ whose head atom matches F (line 15), we instantiate the body atoms of r (line 16), and we recursively examine each instantiated body atom of r (line 18). As soon as F becomes proved (which can happen after a recursive call in line 18), we can stop enumerating supporting facts (lines 14 and 19). Furthermore, a match τ of the body atoms of r where some $\mathsf{b}_i(r)\tau$ has been disproved (i.e., $\mathsf{b}_i(r)\tau \in S$) cannot contribute to a proof of F from the updated set E; therefore, in line 16 we match the body of r in $[I \setminus S]$, rather than in I.

Algorithm 3 implements forward chaining using rules (7) and (8). Lines 20–22 capture rules (7): we extract a fact F from C and add it to P if $F \in E$ holds. Lines 23–29 capture rules (8): we extract a fact from P, identify each relevant rule $r \in \Pi$ (line 25), match its body atoms (line 26), instantiate H as $\mathsf{h}(r)\tau$ (line 27), and add H to P (line 28); to prevent repeated derivations, we accumulate all extracted facts in set V (line 24) and use V to match the body atoms of rules (line 26). Moreover, the body of rule (8) is 'guarded' (i.e., H should be added to P only if $H \in C$ holds) to prevent proving facts not in C. Thus, if $H \notin C$ holds in line 27, we add H to a set Y of *delayed* facts (line 29); then, if H later gets added to C, we add H to P in line 22.

Theorem 1, whose proof is given in the appendix, shows that the B/F algorithm correctly incrementally updates I, and that Algorithms 1 and 3 do not repeat derivations.

Theorem 1. *When applied to a program Π, sets of facts E and E^-, and set $I := \Pi^\infty(E)$, Algorithm 1 terminates and updates I to $\Pi^\infty(E \setminus E^-)$. Each combination of r and τ is considered at most once, in line 9 or 27, but not both.*

Various sets used in the B/F algorithm can be efficiently implemented by associating with each fact $F \in I$ a bit-mask that determines whether F is contained in a particular set. Then, an annotated query Q can be evaluated in these sets by evaluating Q in I and skipping over facts as appropriate.

To compare B/F with DRed, let M_d be the number of derivations from initial materialisation. By the last point of Theorem 1, during deletion propagation and forward chaining combined the B/F algorithm performs at most M_d derivations, and it can also explore M_d rule instances during backward chaining. In contrast, the DRed algorithm performs at most M_d derivations in step **DR2**, and also at most M_d derivations during steps **DR4** and **DR5** combined. Both algorithms can thus perform at most $2M_d$ derivations, but what constitutes a worst case is quite different: we expect B/F to efficiently delete facts with shallow and/or multiple proofs, whereas we expect DRed to efficiently delete facts not participating in many derivations. Finally, consider the limit when all explicit facts are deleted. Then, DRed performs M_d derivations in step **DR2**, and then it tries to reprove (unsuccessfully) all facts in step **DR4**. In contrast, B/F performs M_d derivations during deletion propagation (just like DRed), but it also explores at most M_d rule instances during backward chaining (it need not explore *all* of them because disproved facts are excluded in line 16); the latter is likely to be less efficient than step **DR4** of DRed.

5 Evaluation

Goals We evaluated our approach against three goals. Our first goal was to compare the performance of the B/F and the DRed algorithms. To focus on 'core differences', we optimised DRed to eliminate redundant derivations as we discussed in Section 3. In addition to comparing the running times, we also compared the 'overall work' in an implementation-independent way by counting the facts that each algorithm examines (i.e., the checked facts in B/F and the overdeleted facts in DRed) and counting the derivations (i.e., rule applications) that the algorithms perform at various stages. Fact derivation is not cheap due to index lookup and it constitutes the bulk of the 'work' in both cases.

Our second goal was motivated by an observation that, as $|E^-|$ increases, computing $\Pi^\infty(E \setminus E^-)$ from scratch becomes easier, whereas incremental update becomes harder; hence, there is always point where the former becomes more efficient than the latter. Thus, to estimate practicality of our approach, we identified an 'equilibrium' point E^-_{eq} at which materialisation from scratch takes roughly the same time as the B/F algorithm; we hoped to show that our algorithm can handle E^- of nontrivial size (compared to E).

Given such an 'equilibrium' point E^-_{eq}, our third goal was to investigate how the two algorithms perform on subsets of E^-_{eq} of various size. This is interesting because incremental update algorithms are often used for small sets E^-.

Selecting Facts to Delete As we have already suggested, the choice of the facts in E^- can significantly affect the performance of the two algorithms. Thus, to estimate typical performance, we would have to run the algorithms on many random subsets of E, which is infeasible since each run can take a long time. However, we expect the 'equilibrium' point E^-_{eq} to be sufficiently large to contain a mixture of 'easy' and 'hard' facts, so the performance of the two algorithms should not be greatly influenced by the actual choices. Consequently, we randomly selected just one 'equilibrium' point E^-_{eq}. Furthermore, for practical reasons, we randomly selected subsets of E^-_{eq} containing 100 facts, 5000 facts, and 25%, 50%, and 75% of E^-_{eq}. Each of these sets contains the previous one, so our test results for larger inputs are more likely to be indicative of 'average' behaviour.

Test Setting Both algorithms can be used with general datalog, but we do not know of any general datalog datasets. In contrast, the Semantic Web community has produced numerous large, publicly available datasets, and ontology query answering is a prominent datalog application. Hence, we used several real-world and synthetic Semantic Web datasets, and we implemented the two algorithms in RDFox (Motik et al. 2014)—a state-of-the-art, RAM-based system for the management of RDF data. Parallelising our algorithm is out of scope of this paper, so we used the single-threaded mode of RDFox. We used a server with 256 GB of RAM and two Intel Xeon E5-2670 CPUs at 2.60GHz running Fedora release 20, kernel version 3.15.10-200.fc20.x86_64. Our system and datasets are available online.[1]

[1] https://krr-nas.cs.ox.ac.uk/2015/AAAI/RDFox/Incremental

Table 2: Experimental results

Dataset		$\lvert E^- \rvert$	$\lvert I \setminus I' \rvert$	Rematerialise		DRed					B/F				
				Time	Derivations	Time		Derivations			Time		Derivations		
				(s)	Fwd	(s)	$\lvert D \rvert$	DR2	DR4	DR5	(s)	$\lvert C \rvert$	Bwd	Sat	Del Prop
	LUBM-1k-L	**100**	113	139.4	212.5M	0.0	1.0k	1.1k	0.8k	1.0k	0.0	0.5k	0.2k	0.3k	0.2k
$\lvert E \rvert$ = 133.6M		**5.0k**	5.5k	101.8	212.5M	0.2	55.5k	67.2k	46.9k	59.8k	0.2	23.0k	9.3k	13.7k	7.4k
$\lvert I \rvert$ = 182.4M		**2.5M**	2.7M	138.5	208.8M	39.4	10.3M	15.2M	6.6M	11.5M	32.8	10.0M	4.1M	5.6M	3.7M
M_t = 121.5s		**5.0M**	5.5M	91.8	205.0M	54.8	17.8M	26.3M	10.5M	18.9M	62.3	18.8M	7.8M	10.1M	7.5M
M_d = 212.5M		**7.5M**	8.3M	89.2	201.3M	71.5	24.3M	35.5M	13.6M	24.3M	85.4	26.7M	11.0M	14.0M	11.2M
		10.0M	11.0M	99.5	197.5M	127.9	30.0M	43.1M	15.9M	28.1M	102.2	34.1M	14.0M	17.4M	15.0M
	UOBM-1k-Uo	**100**	160	3482.0	3.6G	8797.6	1.8G	2.6G	53.2M	2.6G	5.4	0.8k	0.5k	1.3k	0.5k
$\lvert E \rvert$ = 254.8M		**5.0k**	85.2k	3417.8	3.6G	9539.3	1.8G	2.6G	53.2M	2.6G	28.2	105.9k	17.9k	42.1k	104.1k
$\lvert I \rvert$ = 2.2G		**17.0M**	130.9M	3903.1	3.4G	8934.3	1.8G	2.7G	63.7M	2.5G	988.8	175.8M	47.6M	104.0M	196.7M
M_t = 5034.0s		**34.0M**	269.0M	4084.1	3.2G	9492.5	1.9G	2.8G	68.4M	2.4G	1877.2	340.7M	87.5M	182.3M	401.1M
M_d = 3.6G		**51.0M**	422.8M	4010.0	3.0G	10659.3	1.9G	2.9G	71.5M	2.2G	2772.7	513.7M	125.2M	246.8M	622.0M
		68.0M	581.4M	3981.9	2.8G	11351.6	1.9G	2.9G	73.3M	2.1G	3737.3	687.0M	162.5M	289.5M	848.6M
	Claros-L	**100**	212	62.9	128.6M	0.0	0.8k	1.0k	0.2k	0.5k	0.0	0.6k	0.3k	0.7k	0.5k
$\lvert E \rvert$ = 18.8M		**5.0k**	11.3k	62.8	128.6M	0.4	37.8k	50.7k	10.9k	23.9k	0.4	29.1k	18.8k	35.3k	26.8k
$\lvert I \rvert$ = 74.2M		**0.6M**	1.3M	62.3	125.6M	32.3	4.1M	5.5M	1.1M	2.5M	14.9	3.1M	2.0M	3.6M	3.0M
M_t = 78.9s		**1.2M**	2.6M	61.2	122.6M	53.2	7.8M	10.8M	2.0M	4.8M	33.6	6.1M	3.8M	6.7M	6.0M
M_d = 128.6M		**1.7M**	4.0M	60.5	119.5M	73.6	11.4M	15.9M	2.8M	6.8M	47.8	8.9M	5.6M	9.5M	9.1M
		2.3M	5.5M	60.0	116.3M	91.0	14.8M	20.9M	3.6M	8.6M	60.6	11.7M	7.3M	12.0M	12.3M
	Claros-LE	**100**	0.5k	3992.8	12.6G	0.0	1.3k	2.0k	0.3k	0.9k	0.0	1.0k	0.7k	1.0k	1.1k
$\lvert E \rvert$ = 18.8M		**2.5k**	178.9k	5235.1	12.6G	8077.4	5.5M	11.7G	176.6k	11.7G	10.3	216.4k	161.2k	8.8M	320.0k
$\lvert I \rvert$ = 533.7M		**5.0k**	427.5k	4985.1	12.6G	7628.2	6.0M	11.7G	186.0k	11.7G	16.5	485.6k	369.0k	8.9M	769.3k
M_t = 4024.5s		**7.5k**	609.6k	4855.0	12.6G	7419.1	6.5M	11.7G	193.9k	11.7G	19.5	683.4k	516.8k	9.0M	1.1M
M_d = 12.9G		**10.0k**	780.8k	5621.3	12.6G	7557.9	6.8M	11.7G	207.6k	11.7G	3907.2	6.0M	723.0M	11.7G	16.9M

Test Datasets Table 2 summarises the properties of the four datasets we used in our tests.

LUBM (Guo, Pan, and Heflin 2005) is a well-known RDF benchmark. We extracted the datalog fragment of the LUBM ontology as described by Grosof et al. (2003), providing us with the 'lower bound' on the ontology's consequences. We generated the data for 1000 universities, and we designate the resulting dataset as *LUBM-1k-L*.

UOBM (Ma et al. 2006) extends LUBM. Instead of the datalog fragment of the UOBM ontology, we used the 'upper bound' datalog program by Zhou et al. (2013) that entails the UOBM ontology. This program is interesting because facts in the materialisation often have many derivations, and it was already used by Motik et al. (2014) to test RDFox. The bodies of several rules in the program by Motik et al. (2014) contain redundant atoms; for example, $R(x, y) \wedge R(x, y') \rightarrow A(x)$. This introduces a source of inefficiency that would not allow us to effectively compare the two algorithms, so we removed all such atoms manually obtaining the 'upper bound optimised' program. We generated data for 1000 universities, and we designate the resulting dataset as *UOBM-1k-Uo*.

Claros integrates information about major art collections in museums and research institutes. We used two Claros variants: *Claros-L* uses the datalog subset of the Claros ontology, and *Claros-LE* extends *Claros-L* with several 'hard' rules that lead to multiple derivations of facts.

Motik et al. (2014) describe all of these datasets in more detail. In addition to the above mentioned optimisation of UOBM, in this paper we do not axiomatise *owl:sameAs* as a congruence relation: derivations with *owl:sameAs* tend to proliferate so an efficient incremental algorithm would have to treat this property directly; we leave developing such an extension to our future work. Please note that omitting the axiomatisation of *owl:sameAs* has a significant impact on materialisation times, so the times presented in this paper are not comparable to the ones given by Motik et al. (2014).

Results Table 2 summarises our results. For each dataset, column 'Dataset' shows the numbers of explicit ($|E|$) and implicit ($|I|$) facts, and the time (M_t) and the number of derivations (M_d) for materialisation. Columns $|E^-|$ and $|I \setminus I'|$ show the numbers of removed explicit and implicit facts, respectively. For rematerialisation, we show the time and the number of derivations. For DRed, we show the time, the size of set D after step **DR2**, and the numbers of derivations in steps **DR2**, **DR4**, and **DR5**. Finally, for B/F, we show the time, the size of set C upon algorithm's termination, the number of rule instances considered in backward chaining in line 16, the number of forward derivations in line 27, and the number of forward derivations in line 8.

Discussion As one can see from the table, the B/F algorithm consistently outperformed the DRed algorithm, with minor improvements for *LUBM-1k-L* and *Claros-L*, to up to orders of magnitude for *UOBM-1k-Uo* and *Claros-LE*. In all cases, B/F considered fewer facts for possible deletion than DRed, with the difference reaching an order of magnitude in the case of *Claros-LE*. The reduction in the number of considered facts had a direct impact on the number of forward derivations, which can be seen as 'core operations'

for both algorithms. The correspondence between considered facts and derivations can, however, be far from linear and it heavily depends on the program and the facts deleted.

On *LUBM-1k-L* and *Claros-L*, facts are easy to delete and both algorithms scale well. In contrast, on *UOBM-1k-Uo* and *Claros-LE* scalability deteriorates more abruptly. The problems in these two cases are caused by large cliques of interconnected constants. For example, the UOBM ontology contains a symmetric and transitive relation *:hasSameHomeTownWith*, and *Claros-LE* contains a similar relation *:relatedPlaces*. Let T be such a relation; then, for all constants a and b connected in E via zero or more T-steps, materialisation I contains the direct T-edges between a and b; given n such constants, I contains n^2 edges derived via n^3 derivations, and the latter largely determines the materialisation time. Now consider deleting an explicit fact $T(a, b)$.

Regardless of whether $T(a, b)$ still holds after deletion, the DRed algorithm computes in step **DR2** all consequences of $T(a, b)$, which involve all facts $T(c, d)$ where c and d belong to the clique of a and b; doing so requires a cubic number of derivations. Next, in steps **DR4** and **DR5** the algorithm rederives the parts of the original clique that 'survive' the deletion, and this also requires a number of derivations that is cubic in the 'surviving' clique size(s). Hence, regardless of whether and by how much the clique changes, the DRed algorithm performs as many derivations as during initial materialisation and rematerialisation combined, which explains why the performance of DRed on *UOBM-1k-Uo* and *Claros-LE* is independent of $|E^-|$.

The performance of the B/F algorithm greatly depends on whether $T(a, b)$ holds after deletion. If that is the case, the algorithm can sometimes find an alternative proof quickly; for example, if E also contains $T(b, a)$ and if we explore rule instance $T(a, b) \leftarrow T(b, a)$ first, we can prove $T(a, b)$ in just one step. The effectiveness of this depends on the order in which instantiated rule bodies are considered in line 16, so a good heuristic can considerably improve the performance. Now assume that $T(a, b)$ does not hold after deletion. Then for every two elements c and d that remain in the clique of b after deletion (the case when they remain in the clique of a is symmetric), B/F adds $T(c, d)$ to C. Indeed, to disprove $T(a, b)$, we must examine all rule instances that derive $T(a, b)$; but then, $T(a, b) \leftarrow T(a, c) \wedge T(c, b)$ is one such rule instance, so we add $T(a, c)$ to C; then, to disprove $T(a, c)$, we add $T(c, a)$ to C; and, to disprove $T(c, a)$, we consider $T(c, a) \leftarrow T(c, d) \wedge T(d, a)$ and add $T(c, d)$ to C. Furthermore, $T(d, a)$ and $T(a, c)$ do not hold after deletion, so $T(c, d)$ is added to D during deletion propagation. But then, the algorithm repeats all derivations from initial materialisation during deletion propagation and forward chaining combined, and it repeats some derivations from initial materialisation during backward chaining (due to pruning in line 19, not all derivations are explored necessarily). That is why B/F performs well on *Claros-LE* with $7.5k$ deleted facts, but not with $10.0k$ facts: the *:relatedPlaces*-cliques 'survive' in the former case, but in the latter case one clique shrinks by one vertex. Moreover, the deleted facts touch several cliques, some of which 'survive' even in the $10.0k$ case, so B/F outperforms DRed on this input.

Although the DRed algorithm can outperform the B/F algorithm, the latter will be more efficient when it can easily produce proofs for facts. Our experimental results confirm this conjecture; in fact, the B/F algorithm outperforms the DRed algorithm up to the 'equilibrium' point E^-_{eq} on all datasets; on *Claros-L*, this involves deleting 12% of the explicit facts, which is considerably more than what is usually found in practical incremental update scenarios. Thus, our results suggest that the B/F algorithm can be an appropriate choice in many practical scenarios.

6 Conclusion

We have presented a novel B/F algorithm for incrementally updating datalog materialisations. In our evaluation, B/F consistently outperformed the state of the art DRed algorithm—by several orders of magnitude in some cases. Our algorithm can thus significantly extend the applicability of materialisation techniques, particularly in settings where data is subject to large and/or frequent changes. We see two main challenges for our future work. First, we will develop a parallel version of our algorithm to exploit many cores/CPUs available in modern systems. Second, we will extend the algorithm to handle the *owl:sameAs* property.

Acknowledgements

This work was supported by the EPSRC projects MaSI^3, Score! and DBOnto, and by the EU FP7 project Optique.

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison Wesley.

Barbieri, D. F.; Braga, D.; Ceri, S.; Valle, E. D.; and Grossniklaus, M. 2010. Incremental Reasoning on Streams and Rich Background Knowledge. In *Proc. ESWC*, 1–15.

Bishop, B.; Kiryakov, A.; Ognyanoff, D.; Peikov, I.; Tashev, Z.; and Velkov, R. 2011. OWLIM: A family of scalable semantic repositories. *Semantic Web* 2(1):33–42.

de Kleer, J. 1986. An Assumption-Based TMS. *Artificial Intelligence* 28(2):127–162.

Doyle, J. 1979. A Truth Maintenance System. *Artificial Intelligence* 12(3):231–272.

Grosof, B. N.; Horrocks, I.; Volz, R.; and Decker, S. 2003. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. WWW*, 48–57.

Guo, Y.; Pan, Z.; and Heflin, J. 2005. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics* 3(2–3):158–182.

Gupta, A., and Mumick, I. S. 1995. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin* 18(2):3–18.

Gupta, A.; Katiyar, D.; and Mumick, I. S. 1992. Counting solutions to the View Maintenance Problem. In *Proc. of the Workshop on Deductive Databases*, 185–194.

Gupta, A.; Mumick, I. S.; and Subrahmanian, V. S. 1993. Maintaining Views Incrementally. In *Proc. SIGMOD*, 157–166.

Harrison, J. V., and Dietrich, S. W. 1992. Maintenance of materialized views in a deductive database: An update propagation approach. In *Proc. of the Workshop on Deductive Databases*, 56–65.

Horrocks, I.; Patel-Schneider, P. F.; Boley, H.; Tabet, S.; Grosof, B.; and Dean, M. 2004. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission.

Kazakov, Y., and Klinov, P. 2013. Incremental Reasoning in OWL EL without Bookkeeping. In *Proc. ISWC*, 232–247.

Küchenhoff, V. 1991. On the Efficient Computation of the Difference Between Consecutive Database States. In *Proc. DOOD*, 478–502.

Ma, L.; Yang, Y.; Qiu, Z.; Xie, G. T.; Pan, Y.; and Liu, S. 2006. Towards a Complete OWL Ontology Benchmark. In *Proc. ESWC*, 125–139.

Motik, B.; Cuenca Grau, B.; Horrocks, I.; Wu, Z.; Fokoue, A.; and Lutz, C. 2012. OWL 2: Web Ontology Language Profiles (Second Edition). W3C Recommendation.

Motik, B.; Nenov, Y.; Piro, R.; Horrocks, I.; and Olteanu, D. 2014. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In *Proc. AAAI*, 129–137.

Nicolas, J.-M., and Yazdanian, K. 1983. An outline of bdgen: A deductive dbms. In *Proc. IFIP Congress*, 711–717.

Ognyanov, D., and Velkov, R. 2014. Correcting inferred knowledge for expired explicit knowledge. US Patent App. 13/924,209.

Ren, Y., and Pan, J. Z. 2011. Optimising ontology stream reasoning with truth maintenance system. In *Proc. CIKM*, 831–836.

Staudt, M., and Jarke, M. 1996. Incremental Maintenance of Externally Materialized Views. In *Proc. VLDB*, 75–86.

Stocker, M., and Smith, M. 2008. Owlgres: A Scalable OWL Reasoner. In *Proc. OWLED*, 26–27.

Urbani, J.; Kotoulas, S.; Maassen, J.; van Harmelen, F.; and Bal, H. E. 2012. WebPIE: A Web-scale Parallel Inference Engine using MapReduce. *Journal of Web Semantics* 10:59–75.

Urbani, J.; Margara, A.; Jacobs, C. J. H.; van Harmelen, F.; and Bal, H. E. 2013. Dynamite: Parallel materialization of dynamic RDF data. In *Proc. ISWC*, 657–672.

Urpí, T., and Olivé, A. 1992. A method for change computation in deductive databases. In *Proc. VLDB*, 225–237.

Wu, Z.; Eadon, G.; Das, S.; Chong, E. I.; Kolovski, V.; Annamalai, M.; and Srinivasan, J. 2008. Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In *Proc. ICDE*, 1239–1248.

Zhou, Y.; Cuenca Grau, B.; Horrocks, I.; Wu, Z.; and Banerjee, J. 2013. Making the most of your triple store: query answering in OWL 2 using an RL reasoner. In *Proc. WWW*, 1569–1580.

A Proof of Theorem 1

Let Π be a program, and let U be a set of facts. A *derivation tree* for a fact F from U is a finite tree T such that

- the root of T is labelled with F,
- each leaf of T is labelled with a fact from E, and
- for each nonleaf node of T labelled with fact H' whose n children are labelled with facts $B'_1, \ldots, B'_n$, a rule $r \in \Pi$ of the form (4) and a substitution τ exist such that $H' = H\tau$, and $B'_i = B_i\tau$ for each $1 \leq i \leq n$.

The *height* of T is the length of its longest branch, and T is *smallest* if no derivation tree T' for F from U exists of height smaller than the height of T. The *materialisation* $\Pi^\infty(U)$ of U w.r.t. Π is the smallest set containing each fact F for which there exists a derivation tree from U. This definition of $\Pi^\infty(U)$ is equivalent to the one in Section 2.

In the rest of this section, we fix a datalog program Π and sets of facts E and E^-. Moreover, we set $I := \Pi^\infty(E)$, we set $E' := E \setminus E^-$, and we set $I' := \Pi^\infty(E \setminus E^-)$. Finally, we let Π' be the datalog program containing rule (7) for each relation R in Π, as well as rule (8) for each rule in Π of the form (4). We next recapitulate Theorem 1 and present its proof which, for convenience, we split into several claims. The theorem follows from Claims 1, 5, 6, and 7.

Theorem 1. *When applied to a program Π, sets of facts E and E^-, and set $I := \Pi^\infty(E)$, Algorithm 1 terminates and updates I to $\Pi^\infty(E \setminus E^-)$. Each combination of r and τ is considered at most once, in line 9 or 27, but not both.*

Claim 1. *Algorithm 3 ensures that set P contains all consequences of Π' w.r.t. sets E and C. Furthermore, each combination of r and τ is considered in line 27 at most once.*

Proof (Sketch). Algorithm 3 is a straightforward modification of the materialisation algorithm by Motik et al. (2014) applied to program Π', so this claim can be proved by a straightforward modification of the proof of Theorem 1 by Motik et al. (2014); a minor difference is that Algorithm 3 accumulates the facts extracted from P in a set V in line 24 and then evaluates rule bodies in V in line 26 to not repeat derivations. We omit the details for the sake of brevity. □

We next introduce an important invariant on C and P.

Claim 2. *Assume that Algorithm 2 is applied to some fact F, set of facts S such that $S \cap I' = \emptyset$, and sets of facts C and P that initially satisfy the following invariant ($\Diamond$):*

> *for each fact $G \in C$, either $G \in P$ or, for each derivation tree T for G from E', set C contains the facts labelling all children (if any exist) of the root of T.*

Invariant ($\Diamond$) is preserved by the algorithm's execution.

Proof. The proof is by induction on recursion depth of Algorithm 2. Assume that the algorithm is applied to some F, C, P, and S as in the claim. For the induction base, ($\Diamond$) clearly remains preserved if the algorithm returns in line 12.

For the induction step, the induction hypothesis ensures that invariant ($\Diamond$) holds after each recursive call in line 18 for each fact different from F. Furthermore, if the algorithm

returns in line 14 or 19, invariant ($\Diamond$) clearly also holds for F. Otherwise, consider an arbitrary derivation tree T for F from E', and let $B_1, \ldots, B_n$ be the children (if any exist) of the root of T. Since S does not contain any fact provable from E', we have $B_i \notin S$ for each $1 \leq i \leq n$. Furthermore, by the definition of the derivation trees, there exists a rule $r \in \Pi$ and a substitution τ such that $\mathsf{h}(r)\tau = F$ and $\mathsf{b}_i(r)\tau = B_i$ for each $1 \leq i \leq n = \mathsf{len}(r)$. These r and τ are considered at some point in lines 23 and 25, and so due to the recursive calls in line 18 we have that $B_i \in C$ for each $1 \leq i \leq n$. Thus, invariant ($\Diamond$) is preserved by the algorithm's execution, as required. □

Calls in line 4 ensure another invariant on C, P, and S.

Claim 3. *At any point during execution of Algorithm 1, invariant ($\Diamond$) holds, as well as $S \cap I' = \emptyset$ and $P = C \cap I'$.*

Proof. The proof is by induction on the number of iterations of the loop of Algorithm 1. For the induction base, we have $S = C = P = \emptyset$ in line 1, so the claim holds initially.

For the induction step, assume that the claim holds before the call in line 4. Invariant ($\Diamond$) remains preserved by Claim 2. Moreover, Algorithm 3 is called in line 13 after set C is modified in line 12, so by Claim 1 sets P and C contain the materialisation of Π'. But then, due to the structure of rules (7) and (8), it should be obvious that $P \subseteq C \cap I'$ holds. We prove $P \supseteq C \cap I'$ by induction on the height h of a shortest derivation tree of a fact $F \in (C \cap I')$ from E'.

- If $h = 0$, then $F \in E'$; but then, since $F \in C$, we have $F \in P$ due to rules (7), as required.
- Assume that the claim holds for all facts in $C \cap I'$ with a shortest derivation tree from E' of height at most h, and assume that F has a shortest derivation tree from E' of height $h + 1$. By the definition of a derivation tree, rule $r \in \Pi$, substitution τ, and facts $B_1, \ldots, B_n$ exist such that $\mathsf{h}(r)\tau = F$ and $\mathsf{b}_i(r)\tau = B_i \in I'$ for each $1 \leq i \leq n = \mathsf{len}(r)$. By invariant ($\Diamond$), for each $1 \leq i \leq n$ we have $B_i \in P \cup C$. Now the height of the smallest derivation tree for B_i is at most h, so by induction assumption we have $B_i \in P$. But then, due to $F \in C$, rule (8) in Π' obtained from r ensures that $F \in P$, as required.

Thus, $P = C \cap I'$ holds after line 4. But then, $S \cap I' = \emptyset$ clearly holds after line 5, as required. □

We next show that Algorithm 1 correctly identifies all facts that must be deleted and adds them to set D.

Claim 4. *$I \setminus I' \subseteq D$ holds in line 11 of Algorithm 1.*

Proof. Consider any fact $F \in I \setminus I'$. We prove the claim by induction on the height h of a smallest derivation tree for F from E; at least one exists due to $F \in I$. Clearly, $F \notin E'$.

- If $h = 0$, then $F \in E$ holds; but then, with $F \notin E'$, we have $F \in E^-$. Thus, we have $F \in D$ due to line 2.
- Assume that the claim holds for all facts in $I \setminus I'$ with a shortest derivation tree from E of height at most h, and assume that F has a shortest derivation tree T from E of height $h + 1$. Let $B_1, \ldots, B_n$ be the facts labelling the children of the root of T, and let $r \in \Pi$ be the rule and let τ be the substitution such that $\mathsf{h}(r)\tau = F$ and $\mathsf{b}_i(r)\tau = B_i$ for $1 \leq i \leq n = \mathsf{len}(r)$. Moreover, let $N = \{B_1, \ldots, B_n\} \setminus I'$; clearly, $N \neq \emptyset$ because otherwise we would have $F \in I'$. Each element of N has a smallest derivation tree from E of height at most h, so by induction assumption we have $N \subseteq D$. Now let $B \in N$ be the fact that is extracted in line 3 before any other fact in $N \setminus \{B\}$; at the point when B is extracted, we have $B \notin O$ (since B is added to O only in line 10). Furthermore, we have $B \notin P$ since B does not have a proof from E', so the check in line 6 passes. Now let $i \in \{1, \ldots, n\}$ be the smallest integer such that $\mathsf{b}_i(r)\tau = B$; then, $\langle r, Q, \sigma \rangle \in \Pi.\mathsf{matchBody}(B)$ is considered in line 7 where Q is the annotated query (9) for the mentioned i. Moreover, the annotations on Q ensure that $\tau \in [I \setminus O].\mathsf{eval}(Q, \{B\}, \sigma)$ is considered in line 8, and so $F = \mathsf{h}(r)\tau$ is added to D in line 9, as required. □

Finally, we show that Algorithm 1 computes the incremental update of I with E^- without repeating derivations, and that derivations are not repeated between lines 9 and 27.

Claim 5. *Algorithm 1 updates set I to I'.*

Proof. Each fact F extracted from D in line 3 is passed in line 4 to Algorithm 2, which in line 12 ensures $F \in C$; consequently, we have $D \subseteq C$ in line 11. Now consider the execution of Algorithm 1 just before line 11. For an arbitrary fact $F \in I \setminus I'$, by Claim 4, we have $F \in D \subseteq C$; by Claim 3 we have $F \notin P$; and so we have $F \in D \setminus P$. Conversely, for an arbitrary fact $F \in D \setminus P$, since $F \in D \subseteq C$, by Claim 3 we have $F \notin I'$; moreover, since we clearly have $D \subseteq I$, we have $F \in I \setminus I'$. We thus have $I \setminus I' = D \setminus P$, so line 11 updates I to I', as required. □

Claim 6. *Each combination of r and τ is considered in line 9 at most once.*

Proof. Assume that a rule $r \in \Pi$ and substitution τ exist that are considered in line 9 twice, when (not necessarily distinct) facts F and F' are extracted from D. Moreover, let B_i and $B_{i'}$ be the body atoms of r that τ matches to F and F'—that is, $F = B_i\tau$ and $F' = B_{i'}\tau$. Finally, let Q' be the annotated query considered in line 7 when atom $B_{i'}$ of r is matched to F'. We have the following possibilities.

- Assume that $F = F'$. Then, B_i and $B_{i'}$ must be distinct, so w.l.o.g. assume that $i \leq i'$. But then, query Q' contains atom $B_i^{\neq}$, so τ cannot be returned in line 8 when evaluating Q'.
- Assume that $F \neq F'$ and that, w.l.o.g. F is extracted from D before F'. Then, we have $F \in O$ due to line 10, and therefore we have $F \notin I \setminus O$; consequently, τ cannot be returned in line 8 when evaluating Q'. □

Claim 7. *Pairs of r and τ that are considered in line 9 are not considered in line 27, and vice versa.*

Proof. For r and τ in line 9, at least some body atom of $r\tau$ does not hold after the update; in contrast, for r and τ in line 27, all body atoms of $r\tau$ hold after the update. □