Optimised Maintenance of Datalog Materialisations

Pan Hu and **Boris Motik** and **Ian Horrocks**
Department of Computer Science, University of Oxford
Oxford, United Kingdom
firstname.lastname@cs.ox.ac.uk

Abstract

To efficiently answer queries, datalog systems often materialise all consequences of a datalog program, so the materialisation must be updated whenever the input facts change. Several solutions to the materialisation update problem have been proposed. The *Delete/Rederive* (DRed) and the *Backward/Forward* (B/F) algorithms solve this problem for general datalog, but both contain steps that evaluate rules 'backwards' by matching their heads to a fact and evaluating the partially instantiated rule bodies as queries. We show that this can be a considerable source of overhead even on very small updates. In contrast, the *Counting* algorithm does not evaluate the rules 'backwards', but it can handle only nonrecursive rules. We present two hybrid approaches that combine DRed and B/F with Counting so as to reduce or even eliminate 'backward' rule evaluation while still handling arbitrary datalog programs. We show empirically that our hybrid algorithms are usually significantly faster than existing approaches, sometimes by orders of magnitude.

1 Introduction

Datalog (Abiteboul, Hull, and Vianu 1995) is a rule language that is widely used in modern information systems. Datalog rules can declaratively specify tasks in data analysis applications (Luteberget, Johansen, and Steffen 2016; Piro et al. 2016), allowing application developers to focus on the objective of the analysis—that is, on specifying *what* needs to be computed rather than *how* to compute it (Markl 2014). Datalog can also capture OWL 2 RL (Motik et al. 2009) ontologies possibly extended with SWRL rules (Horrocks et al. 2004). It is implemented in systems such as WebPIE (Urbani et al. 2012), VLog (Urbani, Jacobs, and Krötzsch 2016), Oracle's RDF Store (Wu et al. 2008), OWLIM (Bishop et al. 2011), and RDFox (Nenov et al. 2015), and it is extensively used in practice.

When performance is critical, datalog systems usually precompute the *materialisation* (i.e., the set of all consequences of a program and the explicit facts) in a preprocessing step so that all queries can later be evaluated directly over the materialisation. Recomputing the materialisation from scratch whenever the explicit facts change can be expensive. Systems thus typically use an *incremental maintenance algorithm*, which aims to avoid repeating most of the work. Fact insertion can be effectively handled using the seminaïve algorithm (Abiteboul, Hull, and Vianu 1995), but deletion is much more involved since one has to check whether deleted facts have derivations that persist after the update. The *Delete/Rederive* (DRed) algorithm (Gupta, Mumick, and Subrahmanian 1993; Staudt and Jarke 1996), the *Backward/Forward* (B/F) algorithm (Motik et al. 2015), and the *Counting* algorithm (Gupta, Mumick, and Subrahmanian 1993) are well-known solutions to this problem.

The DRed algorithm handles deletion by first overdeleting all facts that depend on the removed explicit facts, and then rederiving the facts that still hold after overdeletion. The rederivation stage further involves rederiving all overdeleted facts that have alternative derivations, and then recomputing the consequences of the rederived facts until a fixpoint is reached. The algorithm and its variants have been extensively used in practice (Urbani et al. 2013; Ren and Pan 2011). In contrast to DRed, the B/F algorithm searches for alternative derivations immediately (rather than after overdeletion) using a combination of backward and forward chaining. This makes deletion exact and avoids the potential inefficiency of overdeletion. In practice, B/F often, but not always, outperforms DRed (Motik et al. 2015).

Both DRed and B/F search for derivations of deleted facts by evaluating rules 'backwards': for each rule whose head matches the fact being deleted, they evaluate the partially instantiated rule body as a query; each query answer thus corresponds to a derivation. This has two consequences. First, one can examine rule instances that fire both before and after the update, which is redundant. Second, evaluating rules 'backwards' can be inherently more difficult than matching the rules during initial materialisation: our experiments show that this step can, in some cases, prevent effective materialisation maintenance even for very small updates.

In contrast, the Counting algorithm (Gupta, Mumick, and Subrahmanian 1993) does not evaluate rules 'backwards', but instead tracks the number of distinct derivations of each fact: a counter is incremented when a new derivation for the fact is found, and it is decremented when a derivation no longer holds. A fact can thus be deleted when its counter drops to zero, without the potentially costly 'backward' rule evaluation. The algorithm can also be made optimal in the sense that it considers precisely the rule instances that no longer fire after the update and the rule instances that only fire after the update. The main drawback of Counting is that, unlike DRed and B/F, it is applicable only to nonrecursive rules (Nicolas and Yazdanian 1983). Recursion is a key feature of datalog, allowing one to express common properties such as transitivity. Thus, despite its favourable properties, the Counting algorithm does not provide us with a general solution to the materialisation maintenance problem.

Towards the goal of developing efficient general-purpose maintenance algorithms, in this paper we present two hybrid approaches that combine DRed and B/F with Counting. The former tracks the nonrecursive and the recursive derivations separately, which allows the algorithm to eliminate all 'backward' rule evaluation and also limit overdeletion. The latter tracks nonrecursive derivations only, which eliminates 'backward' rule evaluation for nonrecursive rules; however, recursive rules can still be evaluated 'backwards' to eagerly identify alternative derivations. Both combinations can handle recursive rules, and they exhibit 'pay-as-you-go' behaviour in the sense that they become equivalent to Counting on nonrecursive rules. Apart from the modest cost of maintaining counters, our algorithms never involve more computation steps than their unoptimised counterparts. Thus, our algorithms combine the best aspects of DRed, B/F, and Counting: without incurring a significant cost, they eliminate or reduce 'backward' rule evaluation, are optimal for nonrecursive rules, and can also handle recursive rules.

We have implemented our hybrid algorithms and have compared them with the original DRed and B/F algorithms on several synthetic and real-life benchmarks. Our experiments show that the cost of counter maintenance is negligible, and that our hybrid algorithms typically outperform existing solutions, sometimes by orders of magnitude. Our test system and datasets are available online.[1]

2 Preliminaries

We now introduce datalog with stratified negation. We fix countable, disjoint sets of *constants* and *variables*. A *term* is a constant or a variable. A vector of terms is written $\vec{t}$, and we often treat it as a set. A (*positive*) *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. A term or an atom is *ground* if it does not contain variables. A *fact* is a ground atom, and a *dataset* is a finite set of facts. A rule r has the form

$$B_1 \wedge \cdots \wedge B_m \wedge \text{not}\, B_{m+1} \wedge \cdots \wedge \text{not}\, B_n \rightarrow H,$$

where $m \geq 0$, $n \geq 0$, and B_i and H are atoms. The *head* $\mathsf{h}(r)$ of r is the atom H, the *positive body* $\mathsf{b}^+(r)$ of r is the set of atoms $B_1, \ldots, B_m$, and the *negative body* $\mathsf{b}^-(r)$ of r is the set of atoms $B_{m+1}, \ldots, B_n$. Rule r must be *safe*: each variable occurring in r must occur in a positive body atom.

A *substitution* σ is a mapping of finitely many variables to constants. For α a term, literal, rule, conjunction, or a vector or set thereof, $\alpha\sigma$ is the result of replacing each occurrence of a variable x in α with $\sigma(x)$ (if the latter is defined).

[1] `http://krr-nas.cs.ox.ac.uk/2017/counting/`

A *stratification* λ of a program Π maps each predicate of Π to a positive integer such that, for each rule $r \in \Pi$ with $\mathsf{h}(r) = P(\vec{t})$, (i) $\lambda(P) \geq \lambda(R)$ for each atom $R(\vec{s}) \in \mathsf{b}^+(r)$, and (ii) $\lambda(P) > \lambda(R)$ for each atom $R(\vec{s}) \in \mathsf{b}^-(r)$. Program Π is *stratifiable* if a stratification λ of Π exists. A rule r with $\mathsf{h}(r) = P(\vec{t})$ is *recursive* w.r.t. λ if an atom $R(\vec{s}) \in \mathsf{b}^+(r)$ exists such that $\lambda(P) = \lambda(R)$; otherwise, r is *nonrecursive* w.r.t. λ. For each positive integer s, program $\Pi^s = \{r \in \Pi \mid \lambda(\mathsf{h}(r)) = s\}$ is the *stratum* s of Π, and programs Π^s_{r} and Π^s_{nr} are the recursive and the nonrecursive subsets, respectively, of Π^s. Finally, O^s is the set of all facts that belong to stratum s—that is, $\mathsf{O}^s = \{P(\vec{c}) \mid \lambda(P) = s\}$.

Rule r' is an *instance* of a rule r if a substitution σ exists mapping all variables of r to constants such that $r' = r\sigma$. For I a dataset, the set $\mathsf{inst}_r[I]$ of instances of r obtained by applying a rule r to I, and the set $\Pi[I]$ of facts obtained by applying a program Π to I are defined as follows.

$$\mathsf{inst}_r[I] = \{r\sigma \mid \mathsf{b}^+(r)\sigma \subseteq I \text{ and } \mathsf{b}^-(r)\sigma \cap I = \emptyset\} \quad (1)$$

$$\Pi[I] = \bigcup_{r \in \Pi} \{\mathsf{h}(r') \mid r' \in \mathsf{inst}_r[I]\} \quad (2)$$

We often say that each instance in $\mathsf{inst}_r[I]$ *fires* on I. We are now ready to define the semantics of stratified datalog. Given a dataset E of *explicit facts* and a stratification λ of Π with maximum stratum index number S, we define the following sequence of datasets: let $I^0_\infty = E$; let $I^s_0 = I^{s-1}_\infty$ for index s with $1 \leq s \leq S$; let $I^s_i = I^s_{i-1} \cup \Pi^s[I^s_{i-1}]$ for each integer $i > 0$; and let $I^s_\infty = \bigcup_{i \geq 0} I^s_i$. Set I^S_∞ is called the *materialisation* of Π w.r.t. E and λ. It is well known that I^S_∞ does not depend on λ, so we usually write it as $\mathsf{mat}(\Pi, E)$. In this paper, we consider the problem of maintaining $\mathsf{mat}(\Pi, E)$: given $\mathsf{mat}(\Pi, E)$ and datasets E^- and E^+, our algorithm computes $\mathsf{mat}(\Pi, (E \setminus E^-) \cup E^+)$ incrementally while minimising the amount of work.

3 Motivation and Intuition

As motivation for our work, we next discuss how evaluating rules 'backwards' can be a significant source of inefficiency during materialisation maintenance. We base our discussion on the DRed algorithm for simplicity, but our conclusions apply to the B/F algorithm as well.

3.1 The DRed Algorithm

To make our discussion precise, we first present the DRed algorithm (Gupta, Mumick, and Subrahmanian 1993; Staudt and Jarke 1996). Let Π be a program with a stratification λ, let E be a set of explicit facts, and assume that the materialisation $I = \mathsf{mat}(\Pi, E)$ of Π w.r.t. E has been computed. Moreover, assume that E should be updated by deleting E^- and inserting E^+. The DRed algorithm efficiently modifies the 'old' materialisation I to the 'new' materialisation $I' = \mathsf{mat}(\Pi, (E \setminus E^-) \cup E^+)$ by deleting some facts and adding others; we call such facts *affected* by the update.

Due to the update, some rule instances that fire on I will no longer fire on I', and some rule instances that do not fire on I will fire on I'; we also call such rule instances *affected* by the update. A key problem in materialisation maintenance is to identify the affected rule instances. Clearly, the body of each affected rule instance must contain an affected fact. Based on this observation, the affected rule instances can be efficiently identified by the following generalisation of the operators $\mathsf{inst}_r[I]$ and $\Pi[I]$ from Section 2. In particular, let I^{p}, I^{n}, P, and N be datasets such that $P \subseteq I^{\mathsf{p}}$ and $N \cap I^{\mathsf{n}} = \emptyset$; then, let

$$\begin{aligned}&\mathsf{inst}_r[I^{\mathsf{p}}, I^{\mathsf{n}} \,\vdots\, P, N] \; = \\ &\quad \{r\sigma \mid \mathsf{b}^+(r)\sigma \subseteq I^{\mathsf{p}} \text{ and } \mathsf{b}^-(r)\sigma \cap I^{\mathsf{n}} = \emptyset, \text{ and} \\ &\qquad \mathsf{b}^+(r)\sigma \cap P \neq \emptyset \text{ or } \mathsf{b}^-(r)\sigma \cap N \neq \emptyset\}\end{aligned} \tag{3}$$

and let

$$\Pi[I^{\mathsf{p}}, I^{\mathsf{n}} \,\vdots\, P, N] = \bigcup_{r \in \Pi} \{\mathsf{h}(r') \mid r' \in \mathsf{inst}_r[I^{\mathsf{p}}, I^{\mathsf{n}} \,\vdots\, P, N]\}.$$

Intuitively, the positive and the negative rule atoms are evaluated in I^{p} and I^{n}; sets P and N identify the affected positive and negative facts; $\mathsf{inst}_r[I^{\mathsf{p}}, I^{\mathsf{n}} \,\vdots\, P, N]$ are the affected instances of r; and $\Pi[I^{\mathsf{p}}, I^{\mathsf{n}} \,\vdots\, P, N]$ are the affected consequences of Π. We define $\mathsf{inst}_r[I^{\mathsf{p}}, I^{\mathsf{n}}]$ and $\Pi[I^{\mathsf{p}}, I^{\mathsf{n}}]$ analogously to above, but without the condition '$\mathsf{b}^+(r)\sigma \cap P \neq \emptyset$ or $\mathsf{b}^-(r)\sigma \cap N \neq \emptyset$'. We omit for readability I^{n} whenever $I^{\mathsf{p}} = I^{\mathsf{n}}$, and furthermore we omit N when $N = \emptyset$. Sets $\Pi[I^{\mathsf{p}}, I^{\mathsf{n}}]$ and $\Pi[I^{\mathsf{p}}, I^{\mathsf{n}} \,\vdots\, P, N]$ can be computed efficiently in practice by evaluating the body of each rule $r \in \Pi$ as a conjunctive query and instantiating the head as needed.

Algorithm 1 formalises DRed. The algorithm processes each stratum s and accumulates the necessary changes to I in the set D of *overdeleted* and the set A of *added* facts. The materialisation is updated in line 6, so, prior to that, I and $(I \setminus D) \cup A$ are the 'old' and the 'new' materialisation, respectively. The computation proceeds in three phases.

In the *overdeletion* phase, D is extended with all facts that depend on a deleted fact. In line 8 the algorithm identifies the facts that are explicitly deleted ($E^- \cap \mathsf{O}^s$) or are affected by deletions in the previous strata ($\Pi^s[I \,\vdots\, D \setminus A, A \setminus D]$), and then in lines 9–13 it computes their consequences. It uses a form of the seminaïve strategy, which ensures that each rule instance is considered only once during overdeletion.

In the *one-step rederivation* phase, R is computed as the set of facts that have been overdeleted, but that hold nonetheless. To this end, in line 4 the algorithm considers each fact F in $D \cap \mathsf{O}^s$, and it adds F to R if F is explicit or it is rederived by a rule instance. The latter involves evaluating rules 'backwards': the algorithm identifies each rule $r \in \Pi^s$ whose head can be matched to F, and it evaluates over the 'new' materialisation the body of r as a query with the head variables bound; fact F holds if the query returns at least one answer. As we discuss shortly, this step can be a major source of inefficiency in practice, and the main contribution of this paper is eliminating 'backward' rule evaluation and thus significantly improving the performance.

In the *insertion* step, in line 15 the algorithm combines the one-step rederived facts (R) with the explicitly added facts ($E^+ \cup \mathsf{O}^s$) and the facts added due to the changes in the previous strata ($\Pi^s[(I \setminus D) \cup A \,\vdots\, A \setminus D, D \setminus A]$), and then in lines 16–20 it computes all of their consequences and adds them to A. Again, the seminaïve strategy ensures that each rule instance is considered only once during insertion.

Algorithm 1 $\text{DRED}(\Pi, \lambda, E, I, E^-, E^+)$

1: $D \coloneqq A \coloneqq \emptyset, \quad E^- = (E^- \cap E) \setminus E^+, \quad E^+ = E^+ \setminus E$
2: **for** each stratum index s with $1 \leq s \leq S$ **do**
3:     OVERDELETE
4:     $R \coloneqq \{F \in D \cap \mathsf{O}^s \mid F \in E \setminus E^-$ or there exist $r \in \Pi^s$ and $r' \in \mathsf{inst}_r[I \setminus (D \setminus A), I \cup A]$ with $F = \mathsf{h}(r')\}$
5:     INSERT
6: $E \coloneqq (E \setminus E^-) \cup E^+, \quad I \coloneqq (I \setminus D) \cup A$

7: **procedure** OVERDELETE
8:     $N_D \coloneqq (E^- \cap \mathsf{O}^s) \cup \Pi^s[I \,\vdots\, D \setminus A, A \setminus D]$
9:     **loop**
10:         $\Delta_D \coloneqq N_D \setminus D$
11:         **if** $\Delta_D = \emptyset$ **then break**
12:         $N_D \coloneqq \Pi^s_r[I \setminus (D \setminus A), I \cup A \,\vdots\, \Delta_D]$
13:         $D \coloneqq D \cup \Delta_D$

14: **procedure** INSERT
15:     $N_A \coloneqq R \cup (E^+ \cap \mathsf{O}^s) \cup \Pi^s[(I \setminus D) \cup A \,\vdots\, A \setminus D, D \setminus A]$
16:     **loop**
17:         $\Delta_A \coloneqq N_A \setminus ((I \setminus D) \cup A)$
18:         **if** $\Delta_A = \emptyset$ **then break**
19:         $A \coloneqq A \cup \Delta_A$
20:         $N_A \coloneqq \Pi^s_r[(I \setminus D) \cup A \,\vdots\, \Delta_A]$

3.2 Problems with Evaluating Rules 'Backwards'

The one-step rederivation in line 4 of Algorithm 1 evaluates rules 'backwards'. In this section we present two examples that demonstrate how this can be a major source of inefficiency. Both examples are derived from datasets we used in our empirical evaluation that we present in Section 6; hence, these problems actually arise in practice.

Our discussion depends on several details. In particular, we assume that all facts are indexed so that all facts matching any given atom (possibly containing constants) can be identified efficiently. Moreover, we assume that conjunctive queries corresponding to rule bodies are evaluated left-to-right: for each match of the first conjunct, we partially instantiate the rest of the body and match it recursively. Finally, we assume that query atoms are reordered prior to evaluation to obtain an efficient evaluation plan.

Example 1. *Let Π and E be the program and the dataset as specified in* (4) *and* (5)*, respectively.*

$$R(x, y_1) \wedge R(x, y_2) \rightarrow S(y_1, y_2) \tag{4}$$

$$E = \{R(a_i, b), R(a_i, c_i) \mid 1 \leq i \leq n\} \tag{5}$$

The materialisation $\mathsf{mat}(\Pi, E)$ *consists of E extended with facts $S(b, b)$, $S(b, c_i)$, $S(c_i, b)$, and $S(c_i, c_i)$ for $1 \leq i \leq n$.*

During materialisation, the body of rule (4) *can be evaluated efficiently left-to-right: we match $R(x, y_1)$ to either $R(a_i, b)$ or $R(a_i, c_i)$; this instantiates $R(x, y_2)$ as $R(a_i, y_2)$, and we use the index to find the matching facts $R(a_i, b)$ and $R(a_i, c_i)$. Thus, $R(x, y_1)$ has $2n$ matches, each of which contributes to two matches of $R(x, y_2)$, so the overall cost of rule matching is $O(n)$. The rule body is symmetric, so reordering the body atoms has no effect.*

Now assume that we delete all $R(a_i, c_i)$ with $1 \leq i \leq n$. DRed then overdeletes all $S(b, c_i)$, $S(c_i, b)$ and $S(c_i, c_i)$

facts in lines 8–13, and this can be done efficiently as in the previous paragraph. Next, in one-step rederivation, the algorithm will match these facts to the head of the rule (4) *and obtain queries* $R(x, b) \wedge R(x, c_i)$*,* $R(x, c_i) \wedge R(x, b)$*, and* $R(x, c_i) \wedge R(x, c_i)$*. All but the last of these queries contain atom* $R(x, b)$ *and, no matter how we reorder the body atoms of* (4)*, we have* n *queries where* $R(x, b)$ *is evaluated first. Each of these* n *queries identifies* n *candidate matches* $R(a_i, b)$ *using the index only to find out that the second atom cannot be matched. Thus,* $R(x, b)$ *is matched to* n^2 *facts in total, so the cost of one-step rederivation is* $O(n^2)$*—one degree higher than for materialisation.*

Example 1 shows that evaluating a rule 'backwards' can be inherently more difficult than evaluating it during materialisation, thus giving rise to a dominating source of inefficiency. In fact, evaluating a rule with m body atoms 'backwards' can be seen as answering a query with $m + 1$ atoms, where the head of the rule is an extra query atom; since the number of atoms determines the complexity of query evaluation, this extra atom increases the algorithm's complexity.

Our next example shows that this problem is exacerbated if the space of admissible plans for queries corresponding to rule bodies is further restricted. This is common in systems that provide *built-in functions*. In particular, to facilitate manipulation of concrete values such as strings or integers, datalog systems often allow rule bodies to contain *built-in atoms* of the form $(t \coloneqq \mathit{exp})$, where t is a term and exp is an expression constructed using constants, variables, functions, and operators as usual. For example, a built-in atom can have the form $(z \coloneqq z_1 + z_2)$, and it assigns to z the sum of z_1 and z_2. The set of supported functions vary among implementations, but a common feature is that all values in exp must be bound by prior atoms before the built-in atom can be evaluated. As we show next, this can be problematic.

Example 2. *Let program* Π *consist of rules* (6) *and* (7)*. If we read* $B(s, t, n)$ *as saying that there is an edge from node* s *to node* t *of length* n*, then the program entails* $D(s, n)$ *if there exists a path of length* n *from node* a *to node* s*.*

$$B(a, y, z) \rightarrow D(y, z) \quad (6)$$

$$D(x, z_1) \wedge B(x, y, z_2) \wedge (z \coloneqq z_1 + z_2) \rightarrow D(y, z) \quad (7)$$

Let E *be the dataset as specified below.*

$$E = \{B(a, b_1, 1), B(a, c_i, 1), B(b_i, d_j, 1) \mid 1 \leq i, j \leq n\}$$

During materialisation, rule (6) *first derives* $D(b_1, 1)$ *and all* $D(c_i, 1)$ *with* $1 \leq i \leq n$*, so the cost of this step is* $O(n)$*. Next, atom* $D(x, z_1)$ *in rule* (7) *is matched to* n *facts* $D(c_i, 1)$ *without deriving anything. Atom* $D(x, z_1)$ *is also matched to* $D(b_1, 1)$ *once, so atom* $B(x, y, z_2)$ *is instantiated to* $B(b_1, y, z_2)$ *and matched to* n *facts* $B(b_1, d_j, 1)$*, deriving* n *facts* $D(d_j, 2)$*. Thus, the cost of rule matching is* $O(n)$*.*

Now assume that $B(a, b_1, 1)$ *is deleted. Then,* $D(b_1, 1)$ *and all* $D(d_j, 2)$ *can be efficiently overdeleted as in the previous paragraph, but trying to prove them is much more difficult. Matching each* $D(d_j, 2)$ *to the head of* (6) *produces a query* $B(a, d_j, 2)$*, which does not produce a rule instance. Moreover, matching* $D(d_j, 2)$ *to the head of* (7) *produces a query* $D(x, z_1) \wedge B(x, d_j, z_2) \wedge (2 \coloneqq z_1 + z_2)$*. Now, as we discussed earlier,* z_1 *and* z_2 *must both be bound before we can evaluate the built-in atom* $(2 \coloneqq z_1 + z_2)$*. If we evaluate* $B(x, d_j, z_2)$ *first, then we try* n *facts* $B(b_i, d_j, 1)$ *with* $1 \leq i \leq n$*; for each of them, atom* $D(x, z_1)$ *is instantiated as* $D(b_i, z_1)$ *and is not matched in the surviving facts. In contrast, if we evaluate* $D(x, z_1)$ *first, then we try* n *facts* $D(c_i, 1)$*; for each of them, atom* $B(x, d_j, z_2)$ *is instantiated as* $B(c_i, y, z_2)$ *and is not matched. Thus, regardless of how we reorder the body of* (7)*, the first atom considers a total of* n^2 *facts, so the cost of one-step rederivation is* $O(n^2)$*.*

To overcome this, one might rewrite the built-in atom as $(z_1 \coloneqq z - z_1)$ *or* $(z_2 \coloneqq z - z_2)$ *so that it can be evaluated immediately after* z *and either* z_1 *or* z_2 *are bound. Either way, one-step rederivation still takes* $O(n^2)$ *steps on our example. Also, built-in expressions are often not invertible.*

4 Combining DRed with Counting

We now address the inefficiencies we outlined in Section 3. Towards this goal, in Section 4.1 we first present the intuitions, and then in Section 4.2 we formalise our solution.

4.1 Intuition

As we already mentioned in Section 1, the Counting algorithm (Gupta, Mumick, and Subrahmanian 1993) does not evaluate rules 'backwards'; instead, it tracks the number of derivations of each fact. The main drawback of Counting is that it cannot handle recursive rules. We now illustrate the intuition behind our DRed^c algorithm, which combines DRed with Counting in a way that eliminates 'backward' rule evaluation, while still supporting recursive rules.

The DRed^c algorithm associates with each fact two counters that track the derivations via the nonrecursive and the recursive rules separately. The counters are decremented (resp. incremented) when the associated fact is derived in overdeletion (resp. insertion), which allows for two important optimisations. First, as in the Counting algorithm, the nonrecursive counter always reflects the number of derivations from facts in earlier strata; hence, a fact with a nonzero nonrecursive counter should never be overdeleted because it clearly remains true after the update. This optimisation captures the behaviour of Counting on nonrecursive rules and it also helps limit overdeletion. Second, if we never overdelete facts with nonzero nonrecursive counters, the only way for a fact to still hold after overdeletion is if its recursive counter is nonzero; hence, we can replace 'backward' rule evaluation by a simple check of the recursive counter. Note, however, that the recursive counters can be checked only after overdeletion finishes. This optimisation extends the idea of Counting to recursive rules to completely avoid 'backward' rule evaluation. The following example illustrates these ideas and compares them to DRed.

Example 3. *Let* Π *be the program containing rule* (8)*.*

$$A(x) \wedge B(x, y) \rightarrow A(y) \quad (8)$$

Moreover, let E *be defined as follows:*

$$E = \{A(a), A(b), A(d), B(a, c), B(b, c), B(c, d), B(d, e)\}$$

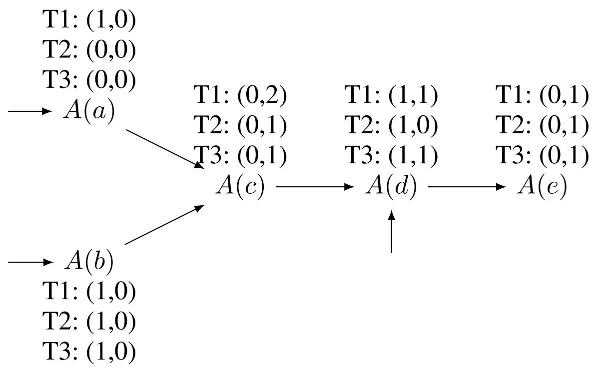

Figure 1: Derivations for Example 3

The materialisation $\mathsf{mat}(\Pi, E)$ *extends* E *with* $A(c)$ *and* $A(e)$*. Figure 1 shows the dependencies between derivations using arrows. For clarity, we do not show the* B*-facts.*

Now assume that $A(a)$ *is deleted. The standard DRed algorithm first overdeletes* $A(a)$*,* $A(c)$*,* $A(d)$*, and* $A(e)$*; it rederives* $A(d)$ *since the fact is in* $E \setminus E^-$*; it rederives* $A(c)$ *by evaluating rule* (8) *'backwards'; and it derives* $A(d)$ *and* $A(e)$ *from the rederived facts.*

Now consider applying the DRed^c *to the same update. For each fact, Figure 1 shows a pair consisting of the nonrecursive and the recursive counter before the update (row T1), after overdeletion (row T2), and after the update (row T3). Note that the presence of a fact in* E *is akin to a nonrecursive derivation, so facts* $A(a)$*,* $A(b)$*, and* $A(d)$ *have nonrecursive derivation counts of one before the update. Now* $A(c)$ *is derived from* $A(a)$ *and* $A(b)$ *using the recursive rule* (8)*, so the recursive counter for* $A(c)$ *is two. Analogously,* $A(d)$ *and* $A(e)$ *have just one recursive derivation each. During overdeletion,* $A(a)$ *is first removed from* E*, so the nonrecursive counter of* $A(a)$ *is decremented to zero and the fact is deleted. Since* $A(a)$ *derives* $A(c)$ *via rule* (8)*, the recursive counter of* $A(c)$ *is decremented; since the nonrecursive counter of* $A(c)$ *is zero, the fact is overdeleted. Since* $A(c)$ *derives* $A(d)$ *via rule* (8)*, the recursive counter of* $A(d)$ *is decremented. Now the nonrecursive counter of* $A(d)$ *is nonzero, so we know that* $A(d)$ *holds after the update; hence, the fact is* not *overdeleted, and the overdeletion phase stops. Thus, while DRed overdeletes four facts, DRed*^{c} *overdeletes only* $A(a)$ *and* $A(c)$*, and does not 'touch'* $A(e)$*.*

Next, DRed^c *proceeds to one-step rederivation. The recursive counter of* $A(c)$ *is nonzero, which means that the fact has a recursive derivation (from* $A(b)$ *in this case) that is not affected. Thus, DRed*^{c} *rederives* $A(c)$ *without any 'backward' rule evaluation.*

Finally, DRed^c *applies insertion. Since* $A(c)$ *derives* $A(d)$ *via* (8)*, the recursive counter of* $A(d)$ *is incremented. Fact* $A(d)$*, however, was not overdeleted, so insertion stops.*

By avoiding 'backward' rule evaluation, DRed^c removes the dominating source of inefficiency on Examples 1 and 2. In fact, on the nonrecursive program from Example 1, the recursive counter is never used and DRed^c performs the same inferences as the Counting algorithm.

4.2 Formalisation

We now formalise our DRed^c algorithm. Our definitions use the standard notion of *multisets*—a generalisation of sets where each element is associated with a positive integer called the *multiplicity* specifying the number of the element's occurrences in the multiset. Moreover, $\oplus$ is the multiset union operator, which adds the elements' multiplicities. If an operand of $\oplus$ is a set, it is treated as a multiset where all elements have multiplicity one. Finally, we extend the notion of rule matching to correctly reflect the number of times a fact is derived: for I^{p}, I^{n}, P, and N datasets with $P \subseteq I^{\mathsf{p}}$ and $N \cap I^{\mathsf{n}} = \emptyset$, we define $\Pi[\![I^{\mathsf{p}}, I^{\mathsf{n}} \mathbin{\vdots} P, N]\!]$ as the multiset containing a distinct occurrence of $\mathsf{h}(r')$ for each rule $r \in \Pi$ and its instance $r' \in \mathsf{inst}_r[I^{\mathsf{p}}, I^{\mathsf{n}} \mathbin{\vdots} P, N]$. This multiset can be computed analogously to $\Pi[I^{\mathsf{p}}, I^{\mathsf{n}} \mathbin{\vdots} P, N]$.

Just like DRed, the DRed^c takes as input a program Π, a stratification λ, a set of explicit facts E and its materialisation $I = \mathsf{mat}(\Pi, E)$, and the sets of facts E^- and E^+ to remove from and add to E. Additionally, the algorithm also takes as input maps C_{nr} and C_{r} that associate each fact F with its nonrecursive and recursive counters $C_{\mathsf{nr}}[F]$ and $C_{\mathsf{r}}[F]$, respectively. These maps should correctly reflect the relevant numbers of derivations. Formally, C_{nr} and C_{r} must be *compatible* with Π, λ, and E, which is the case if $C_{\mathsf{nr}}[F] = C_{\mathsf{r}}[F] = 0$ for each fact $F \notin I$, and, for each fact $F \in I$ and s the stratum index such that $F \in \mathsf{O}^s$ (i.e., s is the index of the stratum that F belongs to),

- $C_{\mathsf{nr}}[F]$ is the multiplicity of F in $E \oplus \Pi^s_{\mathsf{nr}}[\![I]\!]$, and
- $C_{\mathsf{r}}[F]$ is the multiplicity of F in $\Pi^s_{\mathsf{r}}[\![I]\!]$.

For simplicity, we assumes that C_{nr} and C_{r} are defined on all facts, and that $C_{\mathsf{nr}}[F] = C_{\mathsf{r}}[F] = 0$ holds for $F \notin I$; thus, we can simply increment the counters for each newly derived fact in procedure INSERT. In practice, however, one can maintain counters only for the derived facts and initialise the counters to zero for the freshly derived facts.

DRed^c is formalised in Algorithm 2. Its structure is similar to DRed, with the following main differences: instead of evaluating rules 'backwards', one-step rederivation simply checks the recursive counters (line 24); a fact is overdeleted only if the nonrecursive derivation counter is zero (line 34); and the derivation counters are decremented in overdeletion (lines 29–32 and 36–37) and incremented in insertion (lines 41–44 and 49–50). The algorithm also accumulates changes to the materialisation in sets D and A by iteratively processing the strata of λ in three phases.

In the overdeletion phase, DRed^c first considers explicitly deleted facts or facts affected by the changes in earlier strata (lines 29–32). This is analogous to line 8 of DRed, but DRed^c must distinguish Π^s_{nr} from Π^s_{r} so it can decrement the appropriate counters. Next, DRed^c identifies the set Δ_D of facts that have not yet been deleted and whose nonrecursive counter is zero (line 34): a fact with a nonzero nonrecursive counter will always be part of the 'new' materialisation. Note that recursive derivations can be cyclic, so we cannot use the recursive counter to further constrain overdeletion at this point. Then, in lines 35–38 the algorithm propagates consequences of Δ_D just like Algorithm 1, with additionally decrementing the recursive counters in line 37.

Algorithm 2 $\text{DRed}^c(\Pi, \lambda, E, I, E^-, E^+, C_{\mathsf{nr}}, C_{\mathsf{r}})$

21: $D := A := \emptyset, \quad E^- = (E^- \cap E) \setminus E^+, \quad E^+ = E^+ \setminus E$
22: **for** each stratum index s with $1 \leq s \leq S$ **do**
23: OVERDELETE
24: $R := \{F \in D \cap \mathsf{O}^s \mid C_{\mathsf{r}}[F] > 0\}$
25: INSERT
26: $E := (E \setminus E^-) \cup E^+, \quad I := (I \setminus D) \cup A$

27: **procedure** OVERDELETE
28: $N_D := \emptyset$
29: **for** $F \in (E^- \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}}[\![I : D \setminus A, A \setminus D]\!]$ **do**
30: $N_D := N_D \cup \{F\}, \quad C_{\mathsf{nr}}[F] := C_{\mathsf{nr}}[F] - 1$
31: **for** $F \in \Pi^s_{\mathsf{r}}[\![I : D \setminus A, A \setminus D]\!]$ **do**
32: $N_D := N_D \cup \{F\}, \quad C_{\mathsf{r}}[F] := C_{\mathsf{r}}[F] - 1$
33: **loop**
34: $\Delta_D := \{F \in N_D \setminus D \mid C_{\mathsf{nr}}[F] = 0\}$
35: **if** $\Delta_D = \emptyset$ **then break**
36: **for** $F \in \Pi^s_{\mathsf{r}}[\![I \setminus (D \setminus A), I \cup A : \Delta_D]\!]$ **do**
37: $N_D := N_D \cup \{F\}, \quad C_{\mathsf{r}}[F] := C_{\mathsf{r}}[F] - 1$
38: $D := D \cup \Delta_D$

39: **procedure** INSERT
40: $N_A := R$
41: **for** $F \in (E^+ \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}}[\![(I \setminus D) \cup A : A \setminus D, D \setminus A]\!]$ **do**
42: $N_A := N_A \cup \{F\}, \quad C_{\mathsf{nr}}[F] := C_{\mathsf{nr}}[F] + 1$
43: **for** $F \in \Pi^s_{\mathsf{r}}[\![(I \setminus D) \cup A : A \setminus D, D \setminus A]\!]$ **do**
44: $N_A := N_A \cup \{F\}, \quad C_{\mathsf{r}}[F] := C_{\mathsf{r}}[F] + 1$
45: **loop**
46: $\Delta_A := N_A \setminus ((I \setminus D) \cup A)$
47: **if** $\Delta_A = \emptyset$ **then break**
48: $A := A \cup \Delta_A$
49: **for** $F \in \Pi^s_{\mathsf{r}}[\![(I \setminus D) \cup A : \Delta_A]\!]$ **do**
50: $N_A := N_A \cup \{F\}, \quad C_{\mathsf{r}}[F] := C_{\mathsf{r}}[F] + 1$

In the one-step rederivation phase, instead of evaluating rules 'backwards', DRed^c just checks the recursive counter of each fact $F \in D \cap \mathsf{O}^s$ (line 24): if $C_{\mathsf{r}}[F] \neq 0$, then some derivations of F were not 'touched' by overdeletion so F holds in the 'new' materialisation. Conversely, if $C_{\mathsf{r}}[F] = 0$, then $F \in D$ guarantees that $C_{\mathsf{nr}}[F] = 0$ holds as well, so F is not one-step rederivable by a rule in Π.

The insertion phase of DRed^c just uses the seminaïve evaluation while incrementing the counters appropriately.

Without recursive rules, DRed^c becomes equivalent to Counting, and it is optimal in the sense that only affected rule instances are considered during the update. Moreover, the computational complexities of both DRed^c and DRed are the same as for the semi-naive materialisation algorithm: ExpTime in combined and PTime in data complexity (Dantsin et al. 2001). Finally, DRed^c never performs more inferences than DRed and is thus more efficient. Theorem 1 shows that our algorithm is correct, and its proof is given in full in Appendix A.

Theorem 1. *Algorithm 2 correctly updates $I = \mathsf{mat}(\Pi, E)$ to $I' = \mathsf{mat}(\Pi, E')$ for $E' = (E \setminus E^-) \cup E^+$, and it updates C_{nr} and C_{r} so they are compatible with Π, λ, and E'.*

5 Combining B/F with Counting

The B/F algorithm by Motik et al. (2015) uses a combination of backward and forward chaining that makes the deletion phase exact. More specifically, when a fact $F \in \Delta_D$ is considered during deletion, the algorithm uses a combination of backward and forward chaining to look for alternative derivations of F, and it deletes F only if no such derivation can be found. Backward chaining allows B/F to be much more efficient than DRed on many datasets, and this is particularly the case if a program contains many recursive rules. Thus, we cannot hope to remove all 'backward' rule evaluation without eliminating the algorithm's main advantage.

Still, there is room for improvement: backward chaining involves 'backward' evaluation of both nonrecursive and recursive rules, and we can use nonrecursive counters to eliminate the former. Algorithm 3 formalises B/F^c—our combination of the B/F algorithm by Motik et al. (2015) with Counting. The main difference to the original B/F algorithm is that B/F^c associates with each fact a nonrecursive counter that is maintained in lines 59–60 and 89–90, and, instead of evaluating nonrecursive rules 'backwards' to explore alternative derivations of a fact, it just checks in line 79 whether the nonrecursive counter is nonzero. We know that a fact holds if its nonrecursive counter is nonzero; otherwise, we apply backward chaining to *recursive rules only*.We next describe the algorithm's steps in more detail.

Procedure DELETEUNPROVED plays an analogous role to the overdeletion step of DRed and DRed^c. The procedure maintains the nonrecursive counter for each fact in the same way as DRed^c, and the main difference is that a fact F is deleted (i.e., added to Δ_D) in line 66 only if no alternative derivation can be found using a combination of backward and forward chaining implemented in functions CHECK and SATURATE. If an alternative derivation is found, F is added to the set P of *proved* facts.

A call to CHECK(F) searches for the alternative derivations of F using backward chaining. The function maintains the set C of *checked* facts, which ensures that each F is checked only once (line 71 and 78). The procedure first calls SATURATE(F) to determine whether F follows from the facts considered thus far; we discuss this step in more detail shortly. If F is not proved, the procedure then examines in lines 73–76 each instance r' of a recursive rule that derives F in the 'old' materialisation, and it tries to prove all body atoms of r' from the current stratum. This involves evaluating rules 'backwards' and, as we already discussed in Section 5, this is the main advantage of the B/F algorithm over DRed on a number of complex inputs. The function terminates once F is successfully proved (line 76).

Set P accumulates facts that are checked and successfully proved, and it is computed in function SATURATE using forward chaining. Given a fact F that is being checked, it first verifies whether F has a nonrecursive derivation. In the original B/F algorithm, this is done by evaluating the nonrecursive rules 'backwards' in the same way as in line 4 of DRed. In contrast, B/F^c avoids this by simply checking whether the nonrecursive counter is nonzero (line 79): if that is the case, then F is known to have nonrecursive derivations and it is added to P via lines 80 and 82. If F is proved, the procedure propagates its consequences (line 80–85). In particular, the procedure ensures that each consequence F' of P, the facts in the 'new' materialisation in the previous strata, and the recursive rules is added to P if $F' \in C$, or is added to the set Y of *delayed* facts if $F' \notin C$. Intuitively, set Y contains facts that are proved but that have not been checked yet. If a fact in Y is checked at a later point, it is proved in line 79 without having to apply the rules again.

Since the deletion step of B/F^c is 'exact' in the sense that it deletes precisely those facts that no longer hold after the update, rederivation is not needed. Thus, DELETEUNPROVED is directly followed by INSERT, which is the same as in DRed and DRed^c, with the only difference that B/F^c maintains only the nonrecursive counters.

Algorithm 3 is correct in the same way as B/F since checking whether a fact has a nonzero nonrecursive counter is equivalent to checking whether a derivation of the fact exists by evaluating nonrecursve rules 'backwards'.

Algorithm 3 B/F$^c(\Pi, \lambda, E, I, E^-, E^+, C_{\mathsf{nr}})$

51: $D := A := \emptyset, \quad E^- = (E^- \cap E) \setminus E^+, \quad E^+ = E^+ \setminus E$
52: **for** each stratum index s with $1 \le s \le S$ **do**
53: $\quad C := P := Y := \emptyset$
54: $\quad$ DELETEUNPROVED
55: $\quad$ INSERT
56: $E := (E \setminus E^-) \cup E^+, \quad I := (I \setminus D) \cup A$

57: **procedure** DELETEUNPROVED
58: $\quad N_D := \emptyset$
59: $\quad$ **for** $F \in (E^- \cap \mathsf{O}^s) \cup \Pi^s_{\mathsf{nr}}[\![I : D \setminus A, A \setminus D]\!]$ **do**
60: $\quad\quad N_D := N_D \cup \{F\}, \quad C_{\mathsf{nr}}[F] := C_{\mathsf{nr}}[F] - 1$
61: $\quad N_D := N_D \cup \Pi^s_{\mathsf{r}}[I : D \setminus A, A \setminus D]$
62: $\quad$ **loop**
63: $\quad\quad \Delta_D := \emptyset$
64: $\quad\quad$ **for** $F \in N_D \setminus D$ **do**
65: $\quad\quad\quad$ CHECK(F)
66: $\quad\quad\quad$ **if** $F \notin P$ **then** $\Delta_D := \Delta_D \cup \{F\}$
67: $\quad\quad$ **if** $\Delta_D = \emptyset$ **then break**
68: $\quad\quad N_D := \Pi^s_{\mathsf{r}}[I \setminus (D \setminus A), I \cup A : \Delta_D]$
69: $\quad\quad D := D \cup \Delta_D$

70: **function** CHECK(F)
71: $\quad$ **if** $F \notin C$ **then**
72: $\quad\quad$ **if** SATURATE(F) $= \mathsf{f}$ **then**
73: $\quad\quad\quad$ **for** each $r \in \Pi^s_{\mathsf{r}}$ and each
$\quad\quad\quad\quad r' \in \mathsf{inst}_r[I \setminus ((D \cup \Delta_D) \setminus A), I \cup A]$ s.t. $\mathsf{h}(r') = F$ **do**
74: $\quad\quad\quad\quad$ **for** $G \in \mathsf{b}^+(r') \cap \mathsf{O}^s$ **do**
75: $\quad\quad\quad\quad\quad$ CHECK(G)
76: $\quad\quad\quad\quad\quad$ **if** $F \in P$ **then return**

77: **function** SATURATE(F)
78: $\quad C := C \cup \{F\}$
79: $\quad$ **if** $F \in Y$ or $C_{\mathsf{nr}}[F] > 0$ **then**
80: $\quad\quad N_P := \{F\}$
81: $\quad\quad$ **loop**
82: $\quad\quad\quad \Delta_P := (N_P \cap C) \setminus P, \quad Y := Y \cup N_P \setminus C$
83: $\quad\quad\quad$ **if** $\Delta_P = \emptyset$ **then return** t
84: $\quad\quad\quad P := P \cup \Delta_P$
85: $\quad\quad\quad N_P := \Pi^s_{\mathsf{r}}[P \cup (\mathsf{O}^{<s} \cap (I \setminus (D \setminus A))), I \cup A : \Delta_P]$
86: $\quad$ **else return** f

87: **procedure** INSERT
88: $\quad N_A := \emptyset$
89: $\quad$ **for** $F \in (E^+ \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}}[\![(I \setminus D) \cup A : A \setminus D, D \setminus A]\!]$ **do**
90: $\quad\quad N_A := N_A \cup \{F\}, \quad C_{\mathsf{nr}}[F] := C_{\mathsf{nr}}[F] + 1$
91: $\quad N_A := N_A \cup \Pi^s_{\mathsf{nr}}[(I \setminus D) \cup A : A \setminus D, D \setminus A]$
92: $\quad$ **loop**
93: $\quad\quad \Delta_A := N_A \setminus ((I \setminus D) \cup A)$
94: $\quad\quad$ **if** $\Delta_A = \emptyset$ **then break**
95: $\quad\quad A := A \cup \Delta_A$
96: $\quad\quad N_A := N_A \cup \Pi^s_{\mathsf{r}}[(I \setminus D) \cup A : \Delta_A]$

6 Evaluation

We have implemented the unoptimised and optimised variants of DRed and B/F and have compared them empirically.

Benchmarks We used the following benchmarks for our evaluation: UOBM (Ma et al. 2006) is a synthetic benchmark that extends the well known LUBM (Guo, Pan, and Heflin 2005) benchmark; Reactome (Croft et al. 2013) models biological pathways of molecules in cells; Uniprot (Bateman et al. 2015) describes protein sequences and their functional information; ChemBL (Gaulton et al. 2011) represents functional and chemical properties of bioactive compounds; and Claros describes archeological artefacts. Each benchmark consists of a set of facts and an OWL 2 DL ontology, which we transformed into datalog programs of different levels of complexity and recursiveness. More specifically, the *upper bound* (U) programs were obtained using the complete but unsound transformation by Zhou et al. (2013), and they entail all consequences of the original ontology but may also derive additional facts. The *recursive* (R) programs were obtained using the sound but incomplete transformation by Kaminski, Nenov, and Grau (2016), and they tend to be highly recursive. For Claros, the *lower bound extended* (LE) program was obtained by manually introducing several 'hard' rules, and it was already used by Motik et al. (2015) to compare DRed with B/F. Finally, to estimate the effect of built-in literals on materialisation maintenance, we developed a new synthetic benchmark SSPE (Single-Source Path Enumeration). Its dataset consists of a randomly generated directed acyclic graph of 100 k nodes and 1 M edges, and its program traverses paths from a single source analogously to rules (6)–(7). All the tested programs are recursive, although the percentage of the recursive rules varies. Table 1 shows the numbers of facts ($|E|$), strata (S), the nonrecursive rules ($|\Pi_{\mathsf{nr}}|$), and the recursive ones ($|\Pi_{\mathsf{r}}|$) for each benchmark.

Test Setup We conducted all experiments on a Dell PowerEdge R720 server with 256GB RAM and two Intel Xeon E5-2670 2.6GHz processors, running Fedora 24, kernel version 4.8.12-200.fc24.x86_64. All algorithms handle insertions using the seminaïve evaluation. The only overhead is in counter maintenance, which we measured during initial materialisation (which also uses seminaïve evaluation). Hence, the main focus of our tests was on comparing the performance of our algorithms on 'small' and 'large' deletions. In both cases, we first materialised the relevant program on the explicit facts, and then we performed the following tests.

To test small deletions, we measured the performance on

Dataset	$\lvert E\rvert$	S	$\lvert\Pi_{\mathsf{nr}}\rvert$	$\lvert\Pi_{\mathsf{r}}\rvert$	DRed^c	DRed	B/F^c	B/F
UOBM-U	254.8 M	5	135	144	179.24	1,185.87	1.18	31.96
UOBM-R	254.8 M	6	164	2,215	0.29	0.56	0.26	0.24
Reactome-U	12.5 M	9	814	28	0.06	1.03	0.05	1.00
Reactome-R	12.5 M	1	0	21,385	26.57	62.46	0.89	0.90
Uniprot-R	123.1 M	5	9,312	2,706	4.31	8.71	4.13	4.36
ChemBL-R	289.2 M	3	1,766	499	7.91	12.22	1.27	1.26
Claros-LE Best	18.8 M	11	1,031	306	0.33	5,788.75	0.43	8.03
Claros-LE Worst					5,720.57	5,759.92	2,802.86	3,227.05
Claros-LE Average					1,143.55	1,741.66	560.61	653.04
SSPE	3.0 M	1	1	1	10.53	1,684.97	247.00	252.97

Table 1: Average running times for deleting 1000 facts (seconds)

Dataset	$\lvert E^-\rvert/\lvert E\rvert$	DRed^c	DRed	B/F^c	B/F	Remat	Remat-1C	Remat-2C
UOBM-U	50%	1.54 k	3.66 k	1.64 k	3.11 k	1.56 k	1.60 k	1.61 k
UOBM-R	36%	3.28 k	5.75 k	2.76 k	2.87 k	4.14 k	4.16 k	4.19 k
Reactome-U	68%	30.70	6.32 k	39.16	6.33 k	30.90	31.23	31.32
Reactome-R	31%	1.07 k	1.78 k	0.92 k	0.93 k	0.91 k	0.91 k	0.92 k
Uniprot-R	47%	1.57 k	3.47 k	1.92 k	2.86 k	1.98 k	1.99 k	2.00 k
ChemBL-R	69%	4.74 k	7.22 k	3.25 k	4.18 k	2.56 k	2.57 k	2.59 k
Claros-LE	8%	5.01 k	17.81 k	3.49 k	16.74 k	3.36 k	3.49 k	3.60 k
SSPE	2%	74.36	7.24 k	7.90 k	7.75 k	68.28	70.18	71.81

Table 2: Running times for handling large deletions (seconds)

ten randomly selected subsets $E^- \subseteq E$ of 1,000 facts. In all apart from Claros-LE, the running times did not depend significantly on the selected subset of E, so in Table 1 we report the average times across all ten runs. On Claros-LE, however, the running times varied significantly, so we report in the table the best, the worst, and the average times.

To test large deletions, we identified the largest subset $E^- \subseteq E$ on which either DRed^c or B/F^c takes roughly the same time as computing the 'new' materialisation from scratch. We measured the performance of all algorithms on E^-, as well as the performance of rematerialisation with no counters (Remat), just the nonrecursive counter (Remat-1C), and both counters (Remat-2C). This test allows us to assess the scalability of our algorithms. Table 2 reports the running times and the percentages of the deleted facts.

Discussion DRed^c outperformed DRed on all inputs for small deletions. In particular, on SSPE, the average running time for DRed drops from 28 minutes to just over 10 seconds. On Reactome-U, the improvement is by several orders of magnitude, albeit unoptimised DRed is already quite efficient. The improvement is also significant in many other cases, including UOBM-U, Reactome-R, and ChemBL-R. In fact, Reactome-U and SSPE exhibit data and rule patterns outlined in Examples 1 and 2, which clearly demonstrates the benefits of eliminating 'backward' rule evaluation. Moreover, the program of Claros-LE contains a symmetric and transitive predicate $relatedPlaces$, so the materialisation contains several large cliques of constants connected by this predicate (Motik et al. 2015). When a fact $relatedPlaces(a, b)$ is overdeleted, the DRed algorithm overdeletes all $relatedPlaces(c, d)$ where c and d belong to the same clique as a and b, which requires a cubic number of derivations. However, DRed^c can sometimes (but not always) prove that $relatedPlaces(a, b)$ holds using the nonrecursive counter; as one can see, this can considerably improve the performance by avoiding costly overdeletion.

B/F^c also outperformed B/F for small deletions in many cases: B/F^c was more than 20 times faster for UOBM-U, Reactome-U, and the 'best' case of Claros-LE, which is in line with our observation that 'backwards' rule evaluation can be quite costly. In contrast, on the highly recursive datasets (i.e., all R-datasets and SSPE), the performance of B/F^c and B/F is roughly the same: the main source of difficulty is due to the recursive rules, whose evaluation is unaffected by the optimisations proposed in this paper.

B/F^c outperformed DRed^c on all datasets but SSPE. This is so because B/F^c eagerly identifies alternative derivations of facts, which is often easy, and is beneficial since it can considerably reduce overdeletion. However, as we discussed earlier in Section 3, backward rule evaluation can be a dominant source of inefficiencty (e.g., on SSPE). In such cases, DRed^c is faster than B/F^c since DRed^c completely eliminates backward rule evaluation, whereas B/F^c only avoids backward evaluation on nonrecursive rules.

The tests for large deletions show that our algorithms can efficiently delete large subsets of the explicit facts on all but two benchmarks: Claros-LE and SSPE. Claros-LE is difficult due to the presence of cliques as explained earlier, and SSPE is difficult because deleting a small percentage of the explicit facts leads to the deletion of about half of the inferred facts. Nevertheless, DRed^c always considerably outperforms DRed; the difference is particularly significant on Reactome-U and SSPE, where DRed^c is several orders of magnitude faster. Similarly, B/F^c consistently outperforms B/F on all cases apart from SSPE, where the latter is due to the overhead of maintaining the counters.

Finally, the rematerialisation times show that counter maintenance incurs only modest overheads: Remat-2C was in the worst case only several percent slower than Remat.

7 Conclusion

We have presented two novel algorithms for the maintenance of datalog materialisations, obtained by combining the well-known DRed and B/F algorithms with Counting. Our evaluation shows that our algorithms are generally more efficient than the original ones, often by orders of magnitude. Our algorithms could handle both small and large updates efficiently, and have thus been shown to be ready for practical use. In future, we shall develop a modular approach to materialisation and its maintenance that tackles the difficult cases such as Claros-LE using reasoning modules that can be 'plugged into' the seminaïve evaluation to handle difficult rule combinations using custom algorithms.

Acknowledgments

This work was supported by the EPSRC projects MaSI^3, DBOnto, and ED^3.

References

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

Bateman, A.; Martin, M.; O'Donovan, C.; Magrane, M.; Apweiler, R.; Alpi, E.; Antunes, R.; Arganiska, J.; Bely, B.; Bingley, M.; et al. 2015. UniProt: a hub for protein information. *Nucleic Acids Research* 43(D1):D204–D212.

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

Croft, D.; Mundo, A. F.; Haw, R.; Milacic, M.; Weiser, J.; Wu, G.; Caudy, M.; Garapati, P.; Gillespie, M.; Kamdar, M. R.; et al. 2013. The Reactome pathway Knowledgebase. *Nucleic acids research* 42(D1):D472–D477.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys* 33(3):374–425.

Gaulton, A.; Bellis, L. J.; Bento, A. P.; Chambers, J.; Davies, M.; Hersey, A.; Light, Y.; McGlinchey, S.; Michalovich, D.; Al-Lazikani, B.; et al. 2011. ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic acids research* 40(D1):D1100–D1107.

Guo, Y.; Pan, Z.; and Heflin, J. 2005. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web* 3(2):158–182.

Gupta, A.; Mumick, I. S.; and Subrahmanian, V. S. 1993. Maintaining Views Incrementally. In *SIGMOD*. ACM.

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

Kaminski, M.; Nenov, Y.; and Grau, B. C. 2016. Datalog rewritability of Disjunctive Datalog programs and non-Horn ontologies. *Artificial Intelligence* 236:90–118.

Luteberget, B.; Johansen, C.; and Steffen, M. 2016. Rule-Based Consistency Checking of Railway Infrastructure Designs. In *iFM*, 491–507.

Ma, L.; Yang, Y.; Qiu, Z.; Xie, G.; Pan, Y.; and Liu, S. 2006. Towards a Complete OWL Ontology Benchmark. *The Semantic Web: Research and Applications* 125–139.

Markl, V. 2014. Breaking the Chains: On Declarative Data Analysis and Data Independence in the Big Data Era. *P-VLDB* 7(13):1730–1733.

Motik, B.; Patel-Schneider, P.; Parsia, B.; Bock, C.; Fokoue, A.; Haase, P.; Hoekstra, R.; Horrocks, I.; Ruttenberg, A.; Sattler, U.; et al. 2009. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C.

Motik, B.; Nenov, Y.; Piro, R.; and Horrocks, I. 2015. Incremental Update of Datalog Materialisation: the Backward/Forward Algorithm. In *AAAI*, 1560–1568.

Nenov, Y.; Piro, R.; Motik, B.; Horrocks, I.; Wu, Z.; and Banerjee, J. 2015. RDFox: A Highly-Scalable RDF Store. In *ISWC*, 3–20.

Nicolas, J.-M., and Yazdanian, K. 1983. An Outline of BDGEN: A Deductive DBMS. In *IFIP Congress*, 711–717.

Piro, R.; Nenov, Y.; Motik, B.; Horrocks, I.; Hendler, P.; Kimberly, S.; and Rossman, M. 2016. Semantic Technologies for Data Analysis in Health Care. In *ISWC*, 400–417.

Ren, Y., and Pan, J. Z. 2011. Optimising Ontology Stream Reasoning with Truth Maintenance System. In *CIKM*, 831–836.

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

Urbani, J.; Kotoulas, S.; Maassen, J.; Van Harmelen, F.; and Bal, H. 2012. WebPIE: A Web-scale Parallel Inference Engine using MapReduce. *JWS* 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 *ISWC*, 657–672.

Urbani, J.; Jacobs, C. J.; and Krötzsch, M. 2016. Column-Oriented Datalog Materialization for Large Knowledge Graphs. In *AAAI*, 258–264.

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 *ICDE*, 1239–1248. IEEE.

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 *Proceedings of the 22nd international conference on World Wide Web*, 1569–1580. ACM.

A Proof of Correctness for Algorithm 2

Theorem 1. *Algorithm 2 correctly updates $I = \mathsf{mat}(\Pi, E)$ to $I' = \mathsf{mat}(\Pi, E')$ for $E' = (E \setminus E^-) \cup E^+$, and it updates C_{nr} and C_{r} so they are compatible with Π, λ, and E'.*

Proof. Let $\ominus$ be the multiset subtraction operator, and let $\mathsf{Occ}(F, M)$ be the multiplicity of F in multiset M. Due to line 21 of Algorithm 2, without loss of generality we assume that $E^- \subseteq E$ and $E^+ \cap E = \emptyset$. Now let $E|_o = E$ and let $I^0|_o = \emptyset$. Moreover, for each $1 \leq s \leq S$, let $I_0^s|_o, I_1^s|_o, \ldots$ be the sequence of sets where $I_0^s|_o = I^{s-1}|_o \cup (E|_o \cap \mathsf{O}^s)$, and for $i > 0$, $I_i^s|_o = I_{i-1}^s|_o \cup \Pi^s[I_{i-1}^s|_o]$. Index k clearly exists at which the sequence reaches the fixpoint (i.e., $I_k^s|_o = I_{k+1}^s|_o$), so let $I^s|_o = I_k^s|_o$. Finally, let $I|_o = I^S|_o$; we clearly have $I|_o = \mathsf{mat}(\Pi, E|_o)$—that is, $I|_o$ is the 'old' materialisation. Now let $E|_n = (E|_o \setminus E^-) \cup E^+$, and let $I_i^s|_n$, $I^s|_n$, and $I|_n$ be defined analogously, so $I|_n$ is the 'new' materialisation.

For each $1 \leq s \leq S$ and each $F \in I|_o \cap \mathsf{O}^s$, let $C_{\mathsf{nr}}[F]|_o = \mathsf{Occ}(F, E|_o \oplus \Pi_{\mathsf{nr}}^s \llbracket I|_o \rrbracket)$ and $C_{\mathsf{r}}[F]|_o = \mathsf{Occ}(F, \Pi_{\mathsf{r}}^s \llbracket I|_o \rrbracket)$; we define $C_{\mathsf{nr}}[F]|_n$ and $C_{\mathsf{r}}[F]|_n$ analogously using $I|_n$ and $E|_n$.

Now consider a run of Algorithm 2 on $I|_o$, E^-, and E^+. Let $D^0 = A^0 = R^0 = \emptyset$, and for each s with $1 \leq s \leq S$, let D^s, A^s, and R^s be the values of D, A, and R, respectively, after the loop in lines 22–25 finishes for stratum index s. Note that during the execution of Algorithm 2, the set I is equal to $I|_o$ up to before line 26. Furthermore, for each fact $F \in I^s|_o \cap \mathsf{O}^s$, let $C_{\mathsf{nr}}[F]|_d$ and $C_{\mathsf{r}}[F]|_d$ be the values of $C_{\mathsf{nr}}[F]$ and $C_{\mathsf{r}}[F]$, respectively, at the point when OVERDELETE finishes for stratum index s; similarly, for each fact $F \in ((I^s|_o \setminus D^s) \cup A^s) \cap \mathsf{O}^s$, let $C_{\mathsf{nr}}[F]|_a$ and $C_{\mathsf{r}}[F]|_a$ be the values of $C_{\mathsf{nr}}[F]$ and $C_{\mathsf{r}}[F]$, respectively, at the point when INSERT finishes for stratum index s.

$$\Pi_{\mathsf{nr}}^s \llbracket I^s|_o \rrbracket = \Pi_{\mathsf{nr}}^s \llbracket I^s|_o \,\vdots\, D^s \setminus A^{s-1}, A^{s-1} \setminus D^s \rrbracket \oplus \Pi_{\mathsf{nr}}^s \llbracket I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1} \rrbracket \tag{9}$$

$$\Pi_{\mathsf{r}}^s \llbracket I^s|_o \rrbracket = \Pi_{\mathsf{r}}^s \llbracket I^s|_o \,\vdots\, D^s \setminus A^{s-1}, A^{s-1} \setminus D^s \rrbracket \oplus \Pi_{\mathsf{r}}^s \llbracket I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1} \rrbracket \tag{10}$$

$$C_{\mathsf{nr}}[F]|_o - C_{\mathsf{nr}}[F]|_d = \mathsf{Occ}(F, (E^- \cap \mathsf{O}^s) \oplus \Pi_{\mathsf{nr}}^s \llbracket I^s|_o \,\vdots\, D^s \setminus A^{s-1}, A^{s-1} \setminus D^s \rrbracket) \quad \text{for each } F \in I^s|_o \cap \mathsf{O}^s \tag{11}$$

$$C_{\mathsf{r}}[F]|_o - C_{\mathsf{r}}[F]|_d = \mathsf{Occ}(F, \Pi_{\mathsf{r}}^s \llbracket I^s|_o \,\vdots\, D^s \setminus A^{s-1}, A^{s-1} \setminus D^s \rrbracket) \quad \text{for each } F \in I^s|_o \cap \mathsf{O}^s \tag{12}$$

$$I^s|_o \setminus I^s|_n \subseteq D^s \subseteq I^s|_o \tag{13}$$

$$\mathsf{O}^s \cap D^s \cap \Pi_{\mathsf{r}}^s[I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}] \subseteq R^s \subseteq I^s|_n \tag{14}$$

$$I^s|_o \cap A^s \subseteq D^s \tag{15}$$

$$(I^s|_o \setminus D^s) \cup A^s = I^s|_n \tag{16}$$

$$C_{\mathsf{nr}}[F]|_a - C_{\mathsf{nr}}[F]|_d = \mathsf{Occ}(F, (E^+ \cap \mathsf{O}^s) \oplus \Pi_{\mathsf{nr}}^s \llbracket (I^s|_o \setminus D^s) \cup A^s \rrbracket \ominus \Pi_{\mathsf{nr}}^s \llbracket I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1} \rrbracket) \quad \text{for each } F \in ((I^s|_o \setminus D^s) \cup A^s) \cap I^s|_o \cap \mathsf{O}^s \tag{17}$$

$$C_{\mathsf{nr}}[F]|_a = \mathsf{Occ}(F, ((E|_o \setminus E^-) \cup E^+) \oplus \Pi_{\mathsf{nr}}^s \llbracket (I^s|_o \setminus D^s) \cup A^s \rrbracket) \quad \text{for each } F \in (((I^s|_o \setminus D^s) \cup A^s) \setminus I^s|_o) \cap \mathsf{O}^s \tag{18}$$

$$C_{\mathsf{r}}[F]|_a - C_{\mathsf{r}}[F]|_d = \mathsf{Occ}(F, \Pi_{\mathsf{r}}^s \llbracket (I^s|_o \setminus D^s) \cup A^s \rrbracket \ominus \Pi_{\mathsf{r}}^s \llbracket I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1} \rrbracket) \quad \text{for each } F \in ((I^s|_o \setminus D^s) \cup A^s) \cap I^s|_o \cap \mathsf{O}^s \tag{19}$$

$$C_{\mathsf{r}}[F]|_a = \mathsf{Occ}(F, \Pi_{\mathsf{r}}^s \llbracket (I^s|_o \setminus D^s) \cup A^s \rrbracket) \quad \text{for each } F \in (((I^s|_o \setminus D^s) \cup A^s) \setminus I^s|_o) \cap \mathsf{O}^s \tag{20}$$

We shall prove that properties (13), (15), and (16) hold for each s with $0 \leq s \leq S$, and that the other properties hold for each s with $1 \leq s \leq S$. Then, property (16) for $s = S$ and line 26 of Algorithm 2 imply that I is correctly updated from $I|_o$ to $I|_n$. Moreover, $C_{\mathsf{nr}}[F]|_o = \mathsf{Occ}(F, E|_o \oplus \Pi_{\mathsf{nr}}^s \llbracket I|_o \rrbracket)$ and properties (16), (9), (11) and (17) for $1 \leq s \leq S$ jointly imply the following:

$$C_{\mathsf{nr}}[F]|_a = C_{\mathsf{nr}}[F]|_o - (C_{\mathsf{nr}}[F]|_o - C_{\mathsf{nr}}[F]|_d) + (C_{\mathsf{nr}}[F]|_a - C_{\mathsf{nr}}[F]|_d) = \mathsf{Occ}(F, E|_n \oplus \Pi_{\mathsf{nr}}^s \llbracket I|_n \rrbracket) = C_{\mathsf{nr}}[F]|_n \quad \text{for each } F \in I^s|_n \cap I^s|_o \cap \mathsf{O}^s \tag{21}$$

In addition, (16) and (18) imply $C_{\mathsf{nr}}[F]|_a = C_{\mathsf{nr}}[F]|_n$ for each $F \in (I^s|_n \setminus I^s|_o) \cap \mathsf{O}^s$. Therefore, we have $C_{\mathsf{nr}}[F]|_a = C_{\mathsf{nr}}[F]|_n$ for each $F \in I^s|_n \cap \mathsf{O}^s$, which means the nonrecursive counts are correctly updated after the execution of the algorithm. $C_{\mathsf{r}}[F]|_a = C_{\mathsf{r}}[F]|_n$ can be shown analogously using properties (16), (10), (12), (19), and (20).

We prove properties (9)–(20) by induction on s. The base case where $s = 0$ is trivial since all relevant sets in (13), (15), and (16) are empty. For the inductive step, we consider an arbitrary s with $1 \leq s \leq S$ such that (13), (15), and (16) hold for $s - 1$, and we show that properties (9)–(20) hold for s. The proof is lengthy so we break it into several claims. $\square$

Claim 2. *Properties (9) and (10) hold.*

Proof. The way sets D and A are constructed ensures that $(D^s \setminus D^{s-1}) \cap \mathsf{O}^{<s} = \emptyset$ and $A^{s-1} \subseteq \mathsf{O}^{<s}$ hold, and so $A^{s-1} \setminus D^s = A^{s-1} \setminus D^{s-1}$ holds as well, which in turn implies $I^s|_o \cup (A^{s-1} \setminus D^s) = I^s|_o \cup (A^{s-1} \setminus D^{s-1})$. Moreover, the induction assumption $D^{s-1} \subseteq I^{s-1}|_o$ ensures $I^s|_o \cup (A^{s-1} \setminus D^{s-1}) = I^s|_o \cup A^{s-1}$. Therefore, we have $I^s|_o \cup (A^{s-1} \setminus D^s) = I^s|_o \cup A^{s-1}$, which together with the definition of rule application ensures the correctness of the two properties. □

Claim 3. *Property (11) holds.*

Proof. Consider an arbitrary $F \in I^s|_o \cap \mathsf{O}^s$, we have two cases here.

If $F \notin (E^- \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}}[\![I^s|_o \vdots D^s \setminus A^{s-1}, A^{s-1} \setminus D^s]\!]$, then the right-hand side of the equation in (11) equals zero. Moreover, Π^s_{nr} contains only nonrecursive rules, and $A^{s-1} \setminus D^s = A^{s-1} \setminus D^{s-1}$ holds for the same reason as explained in the proof of claim 2; thus we have $F \notin (E^- \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}}[\![I^s|_o \vdots D^{s-1} \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]\!]$; but then, line 29 of Algorithm 2 ensures that the nonrecursive count of F is not decremented during the execution of the OVERDELETE procedure, so the left-hand side of the equation is equal to zero as well. Therefore the property holds in this case.

If $F \in (E^- \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}}[\![I^s|_o \vdots D^s \setminus A^{s-1}, A^{s-1} \setminus D^s]\!] = (E^- \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}}[\![I^s|_o \vdots D^{s-1} \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]\!]$, then line 29 and line 30 guarantee that each distinct occurrence of F in $(E^- \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}}[\![I^s|_o \vdots D^{s-1} \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]\!]$ results in decrementing the nonrecursive count of F by one. Thus the property also holds in this case. □

Claim 4. *Property (12) holds.*

Proof. Line 34 and line 38 ensure that Δ_D used in line 36 is different between two iterations of the loop in lines 33-38, so the rule instances considered in line 36 are different between iterations. The total number of these rule instances is finite and is bounded by the number of rule instances in $\bigcup_{r' \in \Pi^s_{\mathsf{r}}} \mathsf{inst}_{r'}[I^s|_o]$. Thus the loop must terminate, and we let T be the total number of iterations. Moreover, for each $1 \le i \le T$, let D^s_i be the value of D at the beginning of the ith iteration of the loop, and let $C_{\mathsf{r}}[F]|^i_d$ be the value of $C_{\mathsf{r}}[F]$ for each $F \in I^s|_o \cap \mathsf{O}^s$ at the same time point. We prove by induction on i that (22) holds for $1 \le i \le T$. Then (22) for $i = T$ and $A^{s-1} \setminus D^{s-1} = A^{s-1} \setminus D^s$ ensure the correctness of property (12).

$$C_{\mathsf{r}}[F]|_o - C_{\mathsf{r}}[F]|^i_d = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o \vdots D^s_i \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]\!]) \text{ for each } F \in I^s|_o \cap \mathsf{O}^s \tag{22}$$

For the base case, consider an arbitrary $F \in I^s|_o \cap \mathsf{O}^s$. It is easy to see that the recursive count of F has never been changed and should be equal to $C_{\mathsf{r}}[F]|_o$ before line 31 of procedure OVERDELETE for stratum s. But then, line 31 and line 32 ensure that if $F \in \Pi^s_{\mathsf{r}}[\![I^s|_o \vdots D^{s-1} \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]\!]$, then each distinct occurrence of F in the multiset results in decrementing the corresponding recursive count by one; moreover, if $F \notin \Pi^s_{\mathsf{r}}[\![I^s|_o \vdots D^{s-1} \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]\!]$, then the corresponding recursive count will not be changed; either way, $D^s_1 = D^{s-1}$ implies $C_{\mathsf{r}}[F]|_o - C_{\mathsf{r}}[F]|^1_d = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o \vdots D^s_1 \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]\!])$, as required.

For the inductive step, assume that (22) holds for $i-1$ where $1 < i \le T$, and consider arbitrary $F \in I^s|_o \cap \mathsf{O}^s$. Lines 34, 36 and 38 jointly imply (23).

$$C_{\mathsf{r}}[F]|^{i-1}_d - C_{\mathsf{r}}[F]|^i_d = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o \setminus (D^s_{i-1} \setminus A^{s-1}), I^s|_o \cup A^{s-1} \vdots D^s_i \setminus D^s_{i-1}]\!]) \tag{23}$$

We now show that the following holds:

$$\begin{aligned}\mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o \vdots D^s_{i-1} \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]\!]) + \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o \setminus (D^s_{i-1} \setminus A^{s-1}), I^s|_o \cup A^{s-1} \vdots D^s_i \setminus D^s_{i-1}]\!]) \\ = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o \vdots D^s_i \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]\!])\end{aligned} \tag{24}$$

Between $\bigcup_{r' \in \Pi^s_{\mathsf{r}}} \mathsf{inst}_{r'}[I^s|_o \vdots D^s_{i-1} \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]$ and $\bigcup_{r' \in \Pi^s_{\mathsf{r}}} \mathsf{inst}_{r'}[I^s|_o \setminus (D^s_{i-1} \setminus A^{s-1}), I^s|_o \cup A^{s-1} \vdots D^s_i \setminus D^s_{i-1}]$ there is no rule instance repetition, so it is sufficient to show that (25) holds.

$$\begin{aligned}\bigcup_{r' \in \Pi^s_{\mathsf{r}}} \mathsf{inst}_{r'}[I^s|_o \vdots D^s_{i-1} \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}] \cup \mathsf{inst}_{r'}[I^s|_o \setminus (D^s_{i-1} \setminus A^{s-1}), I^s|_o \cup A^{s-1} \vdots D^s_i \setminus D^s_{i-1}] \\ = \bigcup_{r' \in \Pi^s_{\mathsf{r}}} \mathsf{inst}_{r'}[I^s|_o \vdots D^s_i \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]\end{aligned} \tag{25}$$

The $\subseteq$ direction of (25) trivially holds. Now consider the $\supseteq$ direction, let r'' be an arbitrary rule instance contained in the right-hand side of (25), then there exists rule $r' \in \Pi^s_{\mathsf{r}}$ such that $r'' \in \mathsf{inst}_{r'}[I^s|_o \vdots D^s_i \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]$ holds. If we have $r'' \in \mathsf{inst}_{r'}[I^s|_o \vdots D^s_{i-1} \setminus A^{s-1}, A^{s-1} \setminus D^{s-1}]$, then clearly r'' is also contained in the left-hand side of (25). Otherwise, r'' has all positive atoms in $I^s|_o$ but no positive atom in $D^s_{i-1} \setminus A^{s-1}$; moreover, all negative body atoms of r'' have to be matched in $I^s|_o \cup (A^{s-1} \setminus D^{s-1}) = I^s|_o \cup A^{s-1}$; furthermore, r'' has at least one positive body atom in $D^s_i \setminus D^s_{i-1}$; therefore, we have $r'' \in \mathsf{inst}_{r'}[I^s|_o \setminus (D^s_{i-1} \setminus A^{s-1}), I^s|_o \cup A^{s-1} \vdots D^s_i \setminus D^s_{i-1}]$, so r'' is contained in the left-hand side of (25) in this case as well.

(23), (24) and the induction assumption that (22) holds for $i-1$ imply the correctness of (22) for i, and this completes our proof. □

Claim 5. *The right-hand inclusion of property (13) holds.*

Proof. For each rule $r \in \Pi^s$, procedure OVERDELETE for stratum index s considers in lines 29, 31, and 36 only instances of r that are contained in $\mathsf{inst}_r[I^s|_o]$, so the facts derived by these rule instances are in $I^s|_o$. Thus, the claim holds by a straightforward induction on the construction of the set D^s. □

Claim 6. *The left-hand inclusion of property (13) holds.*

Proof. We show by induction that (26) holds for each i.

$$I_i^s|_o \setminus I^s|_n \subseteq D^s \tag{26}$$

For the base case, note that $I_0^s|_o = I^{s-1}|_o \cup (E|_o \cap \mathsf{O}^s)$ and that $I^{s-1}|_o \setminus I^{s-1}|_n \subseteq D^{s-1}$ holds for $s-1$ by the induction assumption. Now consider an arbitrary fact $F \in E|_o \cap \mathsf{O}^s$ such that $F \notin I^s|_n$ holds. Then, the latter ensures $F \notin E|_o \setminus E^-$, which implies $F \in E^-$ and so F is added to N_D in line 30. We next show that F is added to D in line 38: $F \notin I^s|_n$ and $I^s|_n = I^s|_n \cup \Pi^s[I^s|_n]$ imply $\mathsf{Occ}(F, \Pi^s_{\mathsf{nr}}[\![I^s|_n]\!]) = 0$, which in turn implies $\mathsf{Occ}(F, \Pi^s_{\mathsf{nr}}[\![I^{s-1}|_n]\!]) = 0$; but then, $I^{s-1}|_o \setminus (D^{s-1} \setminus A^{s-1}) \subseteq (I^{s-1}|_o \setminus D^{s-1}) \cup A^{s-1}$ and the induction assumption $(I^{s-1}|_o \setminus D^{s-1}) \cup A^{s-1} = I^{s-1}|_n$ ensure $\mathsf{Occ}(F, \Pi^s_{\mathsf{nr}}[\![I^{s-1}|_o \setminus (D^{s-1} \setminus A^{s-1})), I^{s-1}|_o \cup A^{s-1}]\!] = 0$; together with (9), (11), and the definition of $C_{\mathsf{nr}}[F]|_o$ this implies $C_{\mathsf{nr}}[F]|_d = 0$, so the condition in line 34 is satisfied. Hence, $F \in D^s$ holds, as required.

For the inductive step, assume that $I_{i-1}^s|_o$ satisfies (26) for $i > 0$, and consider arbitrary $F \in I_i^s|_o \setminus I^s|_n$. If $F \in I_{i-1}^s|_o$, then $F \in D^s$ holds by the induction assumption. Otherwise, there exist a rule $r \in \Pi^s$ and its instance $r' \in \mathsf{inst}_r[I_{i-1}^s|_o]$ such that $F \in \mathsf{h}(r')$. Definition (3) ensures $\mathsf{b}^+(r') \subseteq I_{i-1}^s|_o \subseteq I^s|_o$ and $\mathsf{b}^-(r') \cap I_{i-1}^s|_o = \emptyset$, and $\mathsf{b}^-(r') \subseteq \mathsf{O}^{<s}$ implies $\mathsf{b}^-(r') \cap I^{s-1}|_o = \mathsf{b}^-(r') \cap I^s|_o = \emptyset$. Finally, $F \notin I^s|_n$ implies $r' \notin \mathsf{inst}_r[I^s|_n]$, so by definition (3) we have one of the following two possibilities.

- $\mathsf{b}^+(r') \not\subseteq I^s|_n$. Thus, a fact $G \in \mathsf{b}^+(r')$ exists such that $G \in I_{i-1}^s|_o \setminus I^s|_n$ holds. The induction assumption for (26) implies $G \in D^s$, and $G \notin I^s|_n$ implies $G \notin I^{s-1}|_n$, so the induction assumption for (16) ensures $G \notin A^{s-1}$; hence, $G \in D^s \setminus A^{s-1}$.
- $\mathsf{b}^-(r') \cap I^s|_n = \mathsf{b}^-(r') \cap I^{s-1}|_n \neq \emptyset$. Thus, a fact $G \in \mathsf{b}^-(r')$ exists such that $G \in I^{s-1}|_n \setminus I^{s-1}|_o$ holds; but then, the right-hand inclusion of (13) implies $G \notin D^{s-1}$, and the induction assumption for (16) implies $G \in A^{s-1}$; therefore, we have $G \in A^{s-1} \setminus D^{s-1} = A^{s-1} \setminus D^s$.

Either way, the right-hand side of (11) or (12) is larger than zero. Hence, the count for F is decremented in OVERDELETE and F is added to N_D. Then, in the same way as in the proof of the base case we have $F \in D^s$, as required. □

Claim 7. *The right-hand inclusion of property* (14) *holds.*

Proof. Consider an arbitrary $F \in R^s$. F can only be added to R^s in line 24, so we have $F \in D^s \cap \mathsf{O}^s$ and $C_{\mathsf{r}}[F]|_d > 0$; $F \in D^s \cap \mathsf{O}^s$ and property (13) ensure $F \in I^s|_o \cap \mathsf{O}^s$; but then, properties (12), (10) and $C_{\mathsf{r}}[F]|_o = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o]\!])$ jointly imply $C_{\mathsf{r}}[F]|_d = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]\!])$; now $C_{\mathsf{r}}[F]|_d > 0$ implies $\mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]\!]) > 0$. Next we show that $\Pi^s_{\mathsf{r}}[\![I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]\!] \subseteq \Pi^s_{\mathsf{r}}[\![I^s|_n]\!]$: due to stratification negative body atoms will only be matched against facts from lower strata, so we have $\Pi^s_{\mathsf{r}}[\![I^s|_n]\!] = \Pi^s_{\mathsf{r}}[\![I^s|_n, I^{s-1}|_n]\!]$; moreover, the left-hand inclusion of property (13) implies $I^s|_o \setminus D^s \subseteq I^s|_n$, which together with the induction assumption for property (16) implies $I^s|_o \setminus (D^s \setminus A^{s-1}) \subseteq (I^s|_o \setminus D^s) \cup A^{s-1} \subseteq I^s|_n$; furthermore, the induction assumption for property (16) ensures $I^{s-1}|_n = (I^{s-1}|_o \setminus D^{s-1}) \cup A^{s-1} \subseteq I^s|_o \cup A^{s-1}$; therefore we clearly have $\Pi^s_{\mathsf{r}}[\![I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]\!] \subseteq \Pi^s_{\mathsf{r}}[\![I^s|_n]\!]$. But then, $\mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]\!]) > 0$ implies $\mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_n]\!]) > 0$, so we have $F \in I^s|_n$, as required. □

Claim 8. *The left-hand inclusion of property* (14) *holds.*

Proof. Consider arbitrary $F \in \mathsf{O}^s \cap D^s \cap \Pi^s_{\mathsf{r}}[I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]$, and we show that $F \in R^s$ holds. $F \in \mathsf{O}^s \cap D^s$ implies $C_{\mathsf{r}}[F]|_d = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}[\![I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]\!])$ in the same way as in the proof of claim 7; but then, the fact that $F \in \Pi^s_{\mathsf{r}}[I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]$ holds imply $C_{\mathsf{r}}[F]|_d > 0$, so line 24 of Algorithm 2 ensures that F is added to R^s, as required. □

Claim 9. *Property* (15) *holds.*

Proof. Consider an arbitrary fact $F \in I^s|_o \cap A^s$: if $F \in I^s|_o \cap A^{s-1}$ holds, then the induction assumption ensures that we have $F \in D^{s-1} \subseteq D^s$; if $F \in I^s|_o \cap (A^s \setminus A^{s-1})$ holds, then F is added to Δ_A in line 46 of procedure INSERT for stratum index s, which ensures $F \notin (I^s|_o \setminus D^s) \cup A^{s-1}$; but then, $F \in I^s|_o$ implies $F \in D^s$, as required. □

Claim 10. *The $\subseteq$ direction of property* (16) *holds.*

Proof. We prove by induction on the construction of A in INSERT that $(I^s|_o \setminus D^s) \cup A \subseteq I^s|_n$ holds. We first consider the base case. Set A is equal to A^{s-1} before the loop in lines 45–50; thus, property (16) is equivalent to $(I^s|_o \setminus D^s) \cup A^{s-1} \subseteq I^s|_n$; now $I^s|_o \setminus D^s \subseteq I^s|_n$ is implied by the left-hand inclusion of property (13), whereas $A^{s-1} \subseteq I^s|_n$ holds by the induction assumption.

For the inductive step, we assume that $(I^s|_o \setminus D^s) \cup A \subseteq I^s|_n$ holds, and we consider ways in which Algorithm 2 can add a fact F to A. If $F \in E^+ \cap \mathsf{O}^s$, then $F \in I^s|_n$ clearly holds. Moreover, if $F \in R^s$ holds, then $F \in I^s|_n$ holds by (14). Otherwise, F is derived in line 41, 43, or 49, so a rule $r \in \Pi^s$ and its instance $r' = \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A]$ exist such that $F \in \mathsf{h}(r')$ holds. But then, definition (3) ensures $\mathsf{b}^+(r') \subseteq (I^s|_o \setminus D^s) \cup A \subseteq I^s|_n$ and $\mathsf{b}^-(r') \cap ((I^s|_o \setminus D^s) \cup A) = \emptyset$, which together with $\mathsf{b}^-(r') \subseteq \mathsf{O}^{<s}$ and the induction assumption for (16) implies $\mathsf{b}^-(r') \cap I^s|_n = \mathsf{b}^-(r') \cap I^{s-1}|_n = \emptyset$. Consequently, we have $r' \in \mathsf{inst}_r[I^s|_n]$, so $F \in I^s|_n$ holds, as required. □

Claim 11. *The $\supseteq$ direction of property* (16) *holds.*

Proof. We show by induction that (27) holds for each i.

$$(I^s|_o \setminus D^s) \cup A^s \supseteq I^s_i|_n \tag{27}$$

For the base case, we have $I^s_0|_n = I^{s-1}|_n \cup (E|_n \cap \mathsf{O}^s) = (I^{s-1}|_o \setminus D^{s-1}) \cup A^{s-1} \cup (E|_n \cap \mathsf{O}^s)$ by the induction assumption for (16). $I^{s-1}|_o \setminus D^{s-1} \subseteq I^s|_o \setminus D^s$ and $A^{s-1} \subseteq A^s$ clearly hold. Now consider arbitrary $F \in E|_n \cap \mathsf{O}^s$. If $F \in E^+$, then lines 41, 42, 46, and 48 ensure $F \in (I^s|_o \setminus D^s) \cup A^s$. If $F \in E|_n \setminus E^+ = E|_o \setminus E^-$, then we clearly have $F \in I^s|_o$; but then, $C_{\mathsf{nr}}[F]|_o = \mathsf{Occ}(F, E|_o \oplus \Pi^s_{\mathsf{nr}}[\![I|_o]\!])$ and property (11) imply $C_{\mathsf{nr}}[F]|_d \geq 1$; thus, line 34 ensures $F \notin D^s$.

For the inductive step, assume that $I^s_{i-1}|_n$ satisfies (27) for $i > 0$, and consider arbitrary $F \in I^s_i|_n$. If $F \in I^s_{i-1}|_n$, then (27) holds by the induction assumption. Otherwise, by definition a rule r and its instance $r' \in \mathsf{inst}_r[I^s_{i-1}|_n]$ exist where $\mathsf{h}(r') = F$. Definition (3) and the induction assumption for (27) ensure $\mathsf{b}^+(r') \subseteq I^s_{i-1}|_n \subseteq (I^s|_o \setminus D^s) \cup A^s$. Moreover, definition (3) also ensures $\mathsf{b}^-(r') \cap I^s_{i-1}|_n = \emptyset$, which together with the induction assumption that property (16) holds for $s-1$ and the definition of $I^s_{i-1}|_n$ implies $\mathsf{b}^-(r') \cap ((I^{s-1}|_o \setminus D^{s-1}) \cup A^{s-1}) = \emptyset$; but then, $\mathsf{b}^-(r') \subseteq \mathsf{O}^{<s}$ implies $\mathsf{b}^-(r') \cap ((I^s|_o \setminus D^s) \cup A^s) = \emptyset$. Now we consider the following cases.

- $\mathsf{b}^+(r') \cap (A^s \setminus A^{s-1}) \neq \emptyset$. Facts in $A^s \setminus A^{s-1}$ are added to A via lines 46 and 48, so there is a point in the execution of the algorithm where $\mathsf{b}^+(r') \cap (A^s \setminus A^{s-1}) \cap \Delta_A \neq \emptyset$ holds in line 49 for the last time for Δ_A. Since $\Delta_A \subseteq A$ holds at this point, we clearly have $\mathsf{b}^+(r') \subseteq (I^s|_o \setminus D^s) \cup A$; moreover, $A \subseteq A^s$ and $\mathsf{b}^-(r') \cap ((I^s|_o \setminus D^s) \cup A^s) = \emptyset$ ensure $\mathsf{b}^-(r') \cap ((I^s|_o \setminus D^s) \cup A) = \emptyset$. But then, $r' \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A \because \Delta_A]$ holds; $F = \mathsf{h}(r')$ will be added to N_A in line 50; and lines 46 and 48 ensure $F \in (I^s|_o \setminus D^s) \cup A^s$.
- $\mathsf{b}^+(r') \cap (A^s \setminus A^{s-1}) = \emptyset$, so $\mathsf{b}^+(r') \subseteq (I^s|_o \setminus D^s) \cup A^{s-1}$ holds. We have the following two possibilities.
  - $\mathsf{b}^+(r') \cap (A^{s-1} \setminus D^s)$ or $\mathsf{b}^-(r') \cap (D^s \setminus A^{s-1})$. But then, by the definition (3) of rule matching, we clearly have $r' \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^{s-1} \because A^{s-1} \setminus D^s, D^s \setminus A^{s-1}]$. But then, lines 41–44 ensure that $F = \mathsf{h}(r')$ is added to N_A; and lines 46 and 48 ensure $F \in (I^s|_o \setminus D^s) \cup A^s$.
  - $\mathsf{b}^+(r') \cap (A^{s-1} \setminus D^s) = \mathsf{b}^-(r') \cap (D^s \setminus A^{s-1}) = \emptyset$. But then, $\mathsf{b}^-(r') \cap (D^s \setminus A^{s-1}) = \emptyset$ and $\mathsf{b}^-(r') \cap (I^s|_o \setminus D^s) \cap A^s = \emptyset$ imply $\mathsf{b}^-(r') \cap (I^s|_o \cup A^{s-1}) = \mathsf{b}^-(r') \cap (I^s|_o \cup A^s) = \emptyset$. Moreover, we argue that each fact $G \in \mathsf{b}^+(r') \subseteq (I^s|_o \setminus D^s) \cup A^{s-1}$ satisfies $G \in I^s|_o \setminus (D^s \setminus A^{s-1})$. This clearly holds if $G \in I^s|_o \setminus D^s$. If $G \in A^{s-1}$, then $G \notin A^{s-1} \setminus D^s$ implies $G \in D^s$, which in turn implies $G \in I^s|_o$ by claim 5; thus, $G \in I^s|_o \setminus (D^s \setminus A^{s-1})$ holds. Now, definition (3) ensures $r' \in \mathsf{inst}_r[I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]$, which implies $F = \mathsf{h}(r') \in I^s|_o$. If $F \notin D^s$, then $F \in (I^s|_o \setminus D^s) \cup A^s$ trivially holds. If $F \in D^s$, property (14) implies $F \in R^s$; then, lines 40, 46, and 48 ensure $F \in (I^s|_o \setminus D^s) \cup A^s$.

□

Claim 12. *The following two properties hold.*

$$\begin{aligned} \Pi^s_{\mathsf{nr}}[\![(I^s|_o \setminus D^s) \cup A^s]\!] = {} & \Pi^s_{\mathsf{nr}}[\![I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]\!] \\ & \oplus \Pi^s_{\mathsf{nr}}[\![(I^s|_o \setminus D^s) \cup A^s \because A^s \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}]\!] \end{aligned} \tag{28}$$

$$\begin{aligned} \Pi^s_{\mathsf{r}}[\![(I^s|_o \setminus D^s) \cup A^s]\!] = {} & \Pi^s_{\mathsf{r}}[\![I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]\!] \\ & \oplus \Pi^s_{\mathsf{r}}[\![(I^s|_o \setminus D^s) \cup A^s \because A^s \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}]\!] \end{aligned} \tag{29}$$

Proof. By the definition of $\Pi[\![I^{\mathsf{p}}, I^{\mathsf{n}} \because P, N]\!]$ it is sufficient to show that for each rule $r \in \Pi^s$, properties (30) and (31) hold.

$$\mathsf{inst}_r[I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}] \cap \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s \because A^s \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}] = \emptyset \tag{30}$$

$$\begin{aligned} \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s] = {} & \mathsf{inst}_r[I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}] \\ & \cup \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s \because A^s \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}] \end{aligned} \tag{31}$$

To prove (30), consider an arbitrary rule instance $r' \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s \because A^s \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}]$. By definition (3) we have $\mathsf{b}^+(r') \cap (A^s \setminus D^{s-1}) \neq \emptyset$ or $\mathsf{b}^-(r') \cap (D^{s-1} \setminus A^{s-1}) \neq \emptyset$; now we examine these two cases separately.

For the first case, let F be an arbitrary fact in $\mathsf{b}^+(r') \cap (A^s \setminus D^{s-1})$. Now if $F \in A^{s-1} \setminus D^{s-1}$ holds, then the induction assumption for (15) ensures $F \notin I^{s-1}|_o$, which in turn implies $F \notin I^s|_o \setminus (D^s \setminus A^{s-1})$; if $F \in A^s \setminus A^{s-1}$ holds, then in the same way as in the proof of claim 9 we have $F \notin (I^s|_o \setminus D^s) \cup A^{s-1}$, which implies $F \notin I^s|_o \setminus (D^s \setminus A^{s-1})$; either way, we have $\mathsf{b}^+(r') \not\subseteq I^s|_o \setminus (D^s \setminus A^{s-1})$, so $r' \notin \mathsf{inst}_r[I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]$ holds.

For the second case, $\mathsf{b}^-(r') \cap (D^{s-1} \setminus A^{s-1}) \neq \emptyset$ and the induction assumption for property (13) imply $\mathsf{b}^-(r') \cap I^{s-1}|_o \neq \emptyset$, which clearly implies $\mathsf{b}^-(r') \cap (I^s|_o \cup A^{s-1}) \neq \emptyset$; thus, by definition (3) we have $r' \notin \mathsf{inst}_r[I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]$; this completes our proof for (30).

Next we prove the $\supseteq$ direction of property (31). Consider an arbitrary rule instance r' contained in the right-hand side of (31): if $r' \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s \,\dot{:}\, A^s \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}]$ holds, then by definition (3) we clearly have $r' \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s]$; if we have $r' \in \mathsf{inst}_r[I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]$, then $\mathsf{b}^+(r') \subseteq I^s|_o \setminus (D^s \setminus A^{s-1})$ implies $\mathsf{b}^+(r') \subseteq (I^s|_o \setminus D^s) \cup A^s$; moreover, $\mathsf{b}^-(r') \cap (I^s|_o \cup A^{s-1}) = \emptyset$ and $\mathsf{b}^-(r') \subseteq \mathsf{O}^{<s}$ jointly imply $\mathsf{b}^-(r') \cap ((I^s|_o \setminus D^s) \cup A^s) = \mathsf{b}^-(r') \cap ((I^{s-1}|_o \setminus D^{s-1}) \cup A^{s-1}) = \emptyset$; therefore, $r' \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s]$ holds by definition (3).

Finally, for the $\subseteq$ direction of property (31), consider arbitrary $r' \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s]$. If $\mathsf{b}^+(r') \cap (A^s \setminus D^{s-1}) \neq \emptyset$ or $\mathsf{b}^-(r') \cap (D^{s-1} \setminus A^{s-1}) \neq \emptyset$ holds, then we clearly have $r' \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s \,\dot{:}\, A^s \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}]$ by definition (3). Otherwise, let F be an arbitrary fact in $\mathsf{b}^+(r')$ and let G be an arbitrary fact in $\mathsf{b}^-(r')$, then we have $F \in ((I^s|_o \setminus D^s) \cup A^s) \setminus (A^s \setminus D^{s-1})$ and $G \notin (I^s|_o \setminus D^s) \cup A^s \cup (D^{s-1} \setminus A^{s-1})$. We next show that $F \in I^s|_o \setminus (D^s \setminus A^{s-1})$ and $G \notin I^s|_o \cup A^{s-1}$ hold.

If $F \in (I^s|_o \setminus D^s) \setminus (A^s \setminus D^{s-1})$ holds, then $F \in I^s|_o \setminus (D^s \setminus A^{s-1})$ trivially holds; if $F \in A^s \setminus (A^s \setminus D^{s-1})$ holds, then we have $F \in A^s \cap D^{s-1} = A^{s-1} \cap D^{s-1} = A^{s-1} \cap D^s$, which in turn implies $F \notin D^s \setminus A^{s-1}$; moreover, $F \in D^s$ and property (13) imply $F \in I^s|_o$; therefore, $F \in I^s|_o \setminus (D^s \setminus A^{s-1})$ holds, as required.

$G \notin A^s$ and $G \notin D^{s-1} \setminus A^{s-1}$ jointly imply $G \notin D^{s-1}$; but then, $G \in \mathsf{O}^{<s}$ implies $G \notin D^s$, which together with $G \notin I^s|_o \setminus D^s$ ensures $G \notin I^s|_o$; thus, $G \notin I^s|_o \cup A^{s-1}$ holds, as required.

F and G are chosen arbitrarily, so we have $\mathsf{b}^+(r') \in I^s|_o \setminus (D^s \setminus A^{s-1})$ and $\mathsf{b}^-(r') \notin I^s|_o \cup A^{s-1}$; by definition (3) we have $r' \in \mathsf{inst}_r[I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]$. This completes our proof for (31). □

Claim 13. *Property* (17) *holds.*

Proof. Due to claim 12 it is now sufficient to show that for each $F \in ((I^s|_o \setminus D^s) \cup A^s) \cap I^s|_o \cap \mathsf{O}^s$, the following property holds.

$$C_{\mathsf{nr}}[F]|_a - C_{\mathsf{nr}}[F]|_d = \mathsf{Occ}(F, (E^+ \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}} [\![(I^s|_o \setminus D^s) \cup A^s \,\dot{:}\, A^s \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}]\!]) \tag{32}$$

Π^s_{nr} contains only nonrecursive rules, so it is sufficient to prove

$$C_{\mathsf{nr}}[F]|_a - C_{\mathsf{nr}}[F]|_d = \mathsf{Occ}(F, (E^+ \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}} [\![(I^s|_o \setminus D^s) \cup A^{s-1} \,\dot{:}\, A^{s-1} \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}]\!]), \tag{33}$$

which is ensured by line 41 and line 42 of the algorithm. □

Claim 14. *Property* (18) *holds.*

Proof. $F \notin I^s|_o \cap \mathsf{O}^s$ ensures $C_{\mathsf{nr}}[F]|_d = C_{\mathsf{nr}}[F]|_o = 0$. Moreover, $F \notin \Pi^s_{\mathsf{nr}} [\![I^s|_o]\!]$ and property (9) jointly imply $F \notin \Pi^s_{\mathsf{nr}} [\![I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]\!]$. Now if $F \notin (E^+ \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}} [\![(I^s|_o \setminus D^s) \cup A^{s-1} \,\dot{:}\, A^{s-1} \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}]\!]$, then by property (28) and $F \notin (E|_o \setminus E^-) \cap \mathsf{O}^s$ we clearly have $\mathsf{Occ}(F, ((E|_o \setminus E^-) \cup E^+) \oplus \Pi^s_{\mathsf{nr}} [\![(I^s|_o \setminus D^s) \cup A^s]\!]) = 0$; moreover, line 41 ensures that the nonrecursive count for F is not incremented in this case, so the left-hand side of (18) equals zero as well. Otherwise, each occurrence of F in the multiset $(E^+ \cap \mathsf{O}^s) \oplus \Pi^s_{\mathsf{nr}} [\![(I^s|_o \setminus D^s) \cup A^{s-1} \,\dot{:}\, A^{s-1} \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}]\!]$ results in incrementing the nonrecursive count of F by one; together with $F \notin (E|_o \setminus E^-) \cap \mathsf{O}^s$ and $F \notin \Pi^s_{\mathsf{nr}} [\![I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}]\!]$ this ensures the correctness of the property. □

Claim 15. *Property* (19) *holds.*

Proof. Line 46 and line 48 ensure that Δ_A used in line 49 is different between iterations of the loop in lines 45–50, so the rle instances considered in line 49 are different between iterations. All these rule instances are in $\bigcup_{r' \in \Pi^s_{\mathsf{r}}} \mathsf{inst}_{r'}[(I^s|_o \setminus D^s) \cup A^s]$, which is equal to $\bigcup_{r' \in \Pi^s_{\mathsf{r}}} \mathsf{inst}_{r'}[I^s|_n]$ by property (16). Therefore, the loop will terminate. Now let T be the total number of iterations; moreover, for each $1 \leq i \leq T$, let A^s_i be the value of A at the beginning of the ith iteration of the loop, and let $C_{\mathsf{r}}[F]|^i_a$ be the value of $C_{\mathsf{r}}[F]$ for each $F \in ((I^s|_o \setminus D^s) \cup A^s) \cap I^s|_o \cap \mathsf{O}^s$ at the same time point. We next prove by induction on i that (34) holds for $1 \leq i \leq T$; then (34) for $i = T$ and property (29) ensure the correctness of the claim.

$$C_{\mathsf{r}}[F]|^i_a - C_{\mathsf{r}}[F]|_d = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}} [\![(I^s|_o \setminus D^s) \cup A^s_i \,\dot{:}\, A^s_i \setminus D^{s-1}, D^{s-1} \setminus A^s_i]\!]) \tag{34}$$

For the base case, we have $A^s_1 = A^{s-1}$. Consider arbitrary $F \in ((I^s|_o \setminus D^s) \cup A^s) \cap I^s|_o \cap \mathsf{O}^s$. Lines 43 and 44 ensure that each occurrence of F in $\Pi^s_{\mathsf{r}} [\![(I^s|_o \setminus D^s) \cup A^{s-1} \,\dot{:}\, A^{s-1} \setminus D^s, D^s \setminus A^{s-1}]\!]$ results in incrementing the corresponding recursive count by one. But then, stratification ensures $\Pi^s_{\mathsf{r}}\llbracket(I^s|_o \setminus D^s) \cup A^{s-1} \vdots A^{s-1} \setminus D^s, D^s \setminus A^{s-1}\rrbracket = \Pi^s_{\mathsf{r}}\llbracket(I^s|_o \setminus D^s) \cup A^{s-1} \vdots A^{s-1} \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}\rrbracket$, so (34) holds for $i = 1$, as required.

For the inductive step, assume that (34) holds for $i-1$ where $1 < i \leq T$, and consider arbitrary $F \in ((I^s|_o \setminus D^s) \cup A^s) \cap I^s|_o \cap \mathsf{O}^s$. Lines 46, 48, and 49 jointly imply (35).

$$C_{\mathsf{r}}[F]|^i_a - C_{\mathsf{r}}[F]|^{i-1}_a = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}\llbracket(I^s|_o \setminus D^s) \cup A^s_i \vdots A^s_i \setminus A^s_{i-1}\rrbracket) \tag{35}$$

We now show that the following holds.

$$\begin{aligned}\mathsf{Occ}(F, \Pi^s_{\mathsf{r}}\llbracket(I^s|_o \setminus D^s) \cup A^s_{i-1} \vdots A^s_{i-1} \setminus D^{s-1}, D^{s-1} \setminus A^s_{i-1}\rrbracket + \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}\llbracket(I^s|_o \setminus D^s) \cup A^s_i \vdots A^s_i \setminus A^s_{i-1}\rrbracket)\\ = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}\llbracket(I^s|_o \setminus D^s) \cup A^s_i \vdots A^s_i \setminus D^{s-1}, D^{s-1} \setminus A^s_i\rrbracket)\end{aligned} \tag{36}$$

To this end, it is sufficient to show that for each $r \in \Pi^s_{\mathsf{r}}$, property (37) holds.

$$\begin{aligned}\mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s_{i-1} \vdots A^s_{i-1} \setminus D^{s-1}, D^{s-1} \setminus A^s_{i-1}] \cup \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s_i \vdots A^s_i \setminus A^s_{i-1}]\\ = \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s_i \vdots A^s_i \setminus D^{s-1}, D^{s-1} \setminus A^s_i]\end{aligned} \tag{37}$$

$(A^s_{i-1} \setminus D^{s-1}) \cup (A^s_i \setminus A^s_{i-1}) = A^s_i \setminus D^{s-1}$ and $D^{s-1} \setminus A^s_{i-1} = D^{s-1} \setminus A^s_i \subseteq \mathsf{O}^{<s}$ ensure that the $\subseteq$ direction of (37) holds. To see that the $\supseteq$ direction holds as well, consider arbitrary $r' \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s_i \vdots A^s_i \setminus D^{s-1}, D^{s-1} \setminus A^s_i]$. If $\mathsf{b}^+(r') \cap (A^s \setminus A^{s-1}) \neq \emptyset$, then we clearly have $r' \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s_i \vdots A^s_i \setminus A^s_{i-1}]$. If $\mathsf{b}^+(r') \cap (A^s_i \setminus A^s_{i-1}) = \emptyset$, then definition (3) ensures that we have $\mathsf{b}^+(r') \subseteq ((I^s|_o \setminus D^s) \cup A^s_i) \setminus (A^s_i \setminus A^s_{i-1}) = (I^s|_o \setminus D^s) \cup A^s_{i-1}$. There are two possibilities here: if $\mathsf{b}^+(r') \cap (A^s_i \setminus D^{s-1}) \neq \emptyset$, then $\mathsf{b}^+(r') \cap (A^s_i \setminus A^s_{i-1}) = \emptyset$ implies $\mathsf{b}^+(r') \cap (A^s_{i-1} \setminus D^{s-1}) \neq \emptyset$; if $\mathsf{b}^-(r') \cap (D^{s-1} \setminus A^s_i) \neq \emptyset$, then clearly $\mathsf{b}^-(r') \cap (D^{s-1} \setminus A^s_{i-1}) \neq \emptyset$ holds as well; either way, we have $F \in \mathsf{inst}_r[(I^s|_o \setminus D^s) \cup A^s_{i-1} \vdots A^s_{i-1} \setminus D^{s-1}, D^{s-1} \setminus A^s_{i-1}]$. Therefore, the $\supseteq$ direction of property (37) holds. But then, property (36) holds, which together with (35) and the induction assumption for (34) ensures that (34) holds for i as well. □

Claim 16. *Property* (20) *holds.*

Proof. $F \notin I^s|_o$ implies $C_{\mathsf{r}}[F]|_d = 0$, so we have $C_{\mathsf{r}}[F]|_a = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}\llbracket(I^s|_o \setminus D^s) \cup A^s \vdots A^s \setminus D^{s-1}, D^{s-1} \setminus A^{s-1}\rrbracket)$ in the same way as in the proof for the previous claim. Moreover, $F \notin I^s|_o$ ensures $\mathsf{Occ}(F, \Pi^s_{\mathsf{r}}\llbracket I^s|_o \setminus (D^s \setminus A^{s-1}), I^s|_o \cup A^{s-1}\rrbracket) = 0$. Therefore, by property (29) we have $C_{\mathsf{r}}[F]|_a = \mathsf{Occ}(F, \Pi^s_{\mathsf{r}}\llbracket(I^s|_o \setminus D^s) \cup A^s\rrbracket)$, as required. □