Combining Rewriting and Incremental Materialisation Maintenance for Datalog Programs with Equality

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

Abstract

Materialisation precomputes all consequences of a set of facts and a datalog program so that queries can be evaluated directly (i.e., independently from the program). Rewriting optimises materialisation for datalog programs with equality by replacing all equal constants with a single representative; and incremental maintenance algorithms can efficiently update a materialisation for small changes in the input facts. Both techniques are critical to practical applicability of datalog systems; however, we are unaware of an approach that combines rewriting and incremental maintenance. In this paper we present the first such combination, and we show empirically that it can speed up updates by several orders of magnitude compared to using either rewriting or incremental maintenance in isolation.

1 Introduction

Datalog [Abiteboul et al., 1995] is a declarative, rule-based language that can describe (possibly recursive) data dependencies. It is widely used in applications as diverse as enterprise data management [Aref, 2010] and query answering over ontologies in the OWL 2 RL profile [Motik et al., 2009] extended with SWRL rules [Horrocks et al., 2004].

Querying the set $\Pi^\approx(E)$ of consequences of a set of explicit facts $E$ and a datalog program $\Pi$ is a key service in datalog systems. It can be supported by precomputing and storing $\Pi^\approx(E)$ so that queries can be evaluated directly, without further reference to $\Pi$. Set $\Pi^\approx(E)$ and the process of computing it are called the materialisation of $E$ w.r.t. $\Pi$. This technique is used in the state of the art systems such as "Olygres" [Stocker and Smith, 2008], WebPIE [Urbani et al., 2012], Oracle’s RDF store [Wu et al., 2008], GraphDB (formerly OWLIM) [Bishop et al., 2011], and RDFox [Motik et al., 2014].

Although datalog traditionally employs the unique name assumption (UNA), in some applications uniqueness of identifiers cannot be guaranteed. For example, due to the distribution and the independence of data sources, in the Semantic Web different identifiers are often used to refer to the same domain object. Handling such use cases requires an extension of datalog without UNA, in which one can infer equalities between constants using a special equality predicate $\approx$ that can occur in facts and rule heads. The semantics of $\approx$ can be captured explicitly using rules that axiomatise $\approx$ as a congruence relation; however, this is known to be inefficient when equality is used extensively. Therefore, systems commonly use rewriting [Baader and Nipkow, 1998; Nieuwenhuis and Rubio, 2001]—an optimisation where equal constants are replaced with a canonical representative, and only facts containing such representatives are stored. The benefits of rewriting have been well-documented in practice [Wu et al., 2008; Urbani et al., 2012; Bishop et al., 2011; Motik et al., 2015a].

Moreover, datalog applications often need to handle continuous updates to the set of explicit facts $E$. Rematerialisation (i.e., computing the materialisation from scratch) is often very costly, so incremental maintenance algorithms are often used in practice. Adding facts to $E$ is trivial as one can simply continue from where the initial materialisation has finished; hence, given a materialisation $\Pi^\approx(E)$ of $E$ w.r.t. $\Pi$ and a set of facts $E^-$, the main challenge for an incremental algorithm is to efficiently compute $\Pi^\approx(E \setminus E^-)$. Several such algorithms have already been proposed. Truth maintenance systems [Doyle, 1979; de Kleer, 1986; Goasdoué et al., 2013] track dependencies between facts to efficiently determine whether a fact has a derivation from $E \setminus E^-$, so only facts for which no such derivations exist are deleted. Such approaches, however, store large amounts of auxiliary information and are thus often unsuitable for data-intensive applications. Counting [Nicolas and Yazdanian, 1983; Gupta et al., 1993; Urbani et al., 2013; Goasdoué et al., 2013] stores with each fact $F \in \Pi^\approx(E)$ the number of times $F$ has been derived during initial materialisation, and this number is used to determine when to delete $F$; however, in its basic form counting works only with nonrecursive rules, and a proposed extension to recursive rules requires multiple counts per fact [Dewani et al., 1992], which can be costly. The Delete/Rederive (DRed) algorithm [Gupta et al., 1993] handles recursive rules with no storage overhead: to delete $E^-$ from $E$, the algorithm first overdeletes all consequences of $E^-$ in $\Pi^\approx(E)$ and then rederives all facts provable from $E \setminus E^-$. The Backward/Forward (B/F) algorithm combines backward and forward chaining in a way that outperforms DRed on inputs where facts have many alternative derivations—a common scenario in Semantic Web applications [Motik et al., 2015b].

Combining rewriting and incremental maintenance is difficult due to complex interactions between the two tech-
niques: removing $E^-$ from $E$ may entail retracting equalities, which may (partially) invalidate the rewriting and require the restoration of rewritten facts (see Section 3). To the best of our knowledge, such a combination has not been considered in the literature, and practical systems either use rewriting with rematerialisation, or axiomatisate equality and use incremental maintenance; in either case they give up a technique known to be critical for performance. In this paper we present the B/F$^{\approx}$ algorithm, which combines rewriting with B/F: given a set of facts $E^-$, our algorithm efficiently updates the materialisation of $E$ w.r.t. II computed using the rewriting approach by Motik et al. (2015a). Extensions of datalog with equality are nowadays used mainly for querying RDF data extended with OWL 2 RL ontologies and SWRL rules, so we formalise our algorithm in the framework of RDF; however, our approach can easily be adapted to general datalog.

We have implemented B/F$^{\approx}$ in the open-source RDFox system and have evaluated it on several real-world and synthetic datasets. Our results show that the algorithm indeed combines the best of both worlds, as it is often several orders of magnitude faster than either rematerialisation with rewriting, or B/F with axiomatised equality.

2 Preliminaries

Datalog. A term is a constant $(a, b, A, R, etc.)$ or a variable $(x, y, z, etc.)$. An RDF atom has the form $(t_1, t_2, t_3)$, where $t_1, t_2, t_3$ are terms; an RDF fact (also called a triple) is a variable-free RDF atom; and a dataset is a finite set of facts. A (datalog) rule $r$ is an implication of the form (1), where $H, B_1, \ldots, B_n$ are atoms and each variable occurring in $H$ also occurs in some $B_i$; $h(r) := H$ is the head atom of $r$; each $B_i$ is a body atom of $r$; and $b(r)$ is the set of all body atoms of $r$. A (datalog) program is a finite set of rules.

\[ H \leftarrow B_1 \land \cdots \land B_n \]  

Equation. The constant owl:sameAs (abbreviated $\approx$) can be used to encode equality between constants. For $\alpha$ a term, atom, rule, or a set of these, $\text{voc}(\alpha)$ is the set of all constants in $\alpha$, and $\text{ov}(\alpha)$ is the result of applying a substitution $\sigma$ to $\alpha$. The materialisation $\Pi_\approx(E)$ of a dataset $E$ w.r.t. II is the smallest superset of $E$ containing $h(r)\sigma$ for each rule $r \in II$ and substitution $\sigma$ with $b(r)\sigma \subseteq \Pi_\approx(E)$.

Rewriting. A well-known optimisation of this approach. For $\pi$ a mapping of constants to constants and $\alpha$ a constant, fact, rule, dataset, or substitution, $\pi(\alpha)$ is the result of replacing each constant $c$ in $\alpha$ with $\pi(c)$; such $\pi$ is normal w.r.t. $\pi$ if $\pi(\alpha) = \alpha$; and $\pi(\alpha)$ is the representative of $\alpha$ in $\pi$. For $c$ a constant, let $c^\pi := \{d \mid \pi(d) = c\}$. For a dataset, let $U^\pi := \{(s, p, o) \mid \pi(s), \pi(p), \pi(o) \in U\}$; and, for a fact, let $F^\pi := \{F\}^\pi$. We assume that all constants are totally ordered such that $\approx$ is the smallest constant; then, for $S$ a nonempty set of constants, $\text{min}(S)$ (resp. $\text{max}(S)$) is the smallest (resp. greatest) element of $S$. Let $U$ be a dataset and let $E_c(U) := \{c\} \cup \{d \mid c \approx d \in U\}$; then, the rewriting of $U$ is the pair $(\pi, I)$ such that

1. $\pi(c) = \text{min}(E_c(U))$ for each constant $c$, and
2. $I = \pi(U)$.

Note that $\pi(\approx) = \approx$, that the rewriting is unique for $U$, and that $\Pi_\approx(U)$ if $U$ implies $I^\pi = U$. The r-materialisation of a dataset $E$ w.r.t. II is a program $\pi$ is the rewriting $(\pi, I)$ of the dataset $J = (\Pi \cup \Pi_\approx(E))$. Motik et al. (2015a) show how to answer queries over $J$ by materialising $(\pi, I)$ instead of $J$.

3 Updating R-Materialisation Incrementally

Let $E$ and $E^-$ be datasets, let $E' = E \setminus E^-$, and let $I$ be a program. Moreover, let $J$ (resp. $J'$) be the materialisation of $E$ (resp. $E'$) w.r.t. II $\Pi_\approx$, and let $(\pi, I)$ (resp. $(\pi', I')$) be the r-materialisation of $E$ (resp. $E'$) w.r.t. II. Given $(\pi, I)$, II, and $E^-$, the B/F$^{\approx}$ algorithm computes $(\pi', I')$ efficiently by combining the B/F algorithm by Motik et al. (2015b) for incremental maintenance in datalog without equality with the r-materialisation algorithm by Motik et al. (2015a).

We discuss the intuition in Section 3.1 and some optimisations in Section 3.2, and we formalise the algorithm in Section 3.3.

3.1 Intuition

Main Difficulty. An update may lead to the deletion of equalities, which may require adding facts to $I$. The following example program II and dataset $E$ exhibit such behaviour.

\[ \Pi = \{ y_1 \approx y_2 \leftarrow \langle y_1, R, x \rangle \land \langle y_2, R, x \rangle, \\
\quad y_1 \approx y_2 \leftarrow \langle x, R, y_1 \rangle \land \langle x, R, y_2 \rangle \} \]

\[ E = \{ \{a, R, b\}, \{c, R, d\}, \{a, R, d\} \} \]

\[ I = \{ \{a, R, b\}, a \approx a, R \approx R, b \approx b, \approx \approx \} \]

\[ \pi = \{ a \mapsto a, b \mapsto b, c \mapsto a, d \mapsto b, R \mapsto R, \approx \mapsto \approx \} \]

\[ E^- = \{ \{a, R, d\} \} \]

\[ I' = \{ \{a, R, b\}, a \approx a, R \approx R, b \approx b, \approx \approx, c \approx c, d \approx d \} \]

\[ \pi' = \{ a \mapsto a, b \mapsto b, c \mapsto c, d \mapsto d, R \mapsto R, \approx \mapsto \approx \} \]

Relation $R$ is bijective in II, so $a \approx c \in J$ as both $a$ and $c$ have outgoing R-edges to $d$, and $b \approx d \in J$ as both $b$ and $d$ have incoming R-edges from $a$. By rewriting, we represent each fact $\langle a, R, b \rangle$ from $J$ using a single fact $\langle a, R, b \rangle$, analogously for facts involving $\approx$; thus, instead of 14 facts, we store just five facts. Assume now that we remove $E^-$ from $E$. In $J$ and $J'$ we ascribe no particular meaning to $\approx$, so the monotonicity of datalog ensures $J \subseteq J'$; thus, the B/F algorithm just needs to delete facts that no longer hold.

\[ \text{http://www.cs.ox.ac.uk/isg/tools/RDFox/} \]
However, \( a \approx c \not\in J' \) and \( b \approx d \not\in J' \), so we must update \( \pi \) and extend \( I \) with the facts from \( J' \) that are not represented via \( \pi' \). Thus, in our example, \( J' \) actually contains \( I \).

**Solution Overview.** \( B/F^\infty \) consists of Algorithms 1–7 that follow the same basic idea as \( B/F \); to highlight the differences, lines that exist in \( B/F \) in a modified form are marked with ‘\(*\)’, and new lines and algorithms are marked with ‘\(b\)’.

We initially mark all facts in \( \pi(E^-) \) as ‘doubtful’—that is, we indicate that their truth might change. Next, for each ‘doubtful’ fact \( F \), we determine whether \( F \) is provable from \( E' \) and, if not, we identify the immediate consequences of \( F \) (i.e., the facts in \( J \) that can be derived using \( F \)) and mark them as ‘doubtful’; we know exactly which facts have changed after processing all ‘doubtful’ facts. To check the provability of \( F \), we use backward chaining to identify the facts in \( I \) that can prove \( F \), and we use forward chaining to actually prove \( F \). The latter process also identifies the necessary changes to \( \pi \) and \( I \), which we apply to \((\pi, I)\) in a final step. We next describe the components of \( B/F^\infty \) in more detail.

**Procedure** saturate() is given a dataset \( C \subseteq I \) of checked facts, and it computes the set \( L \) containing each fact \( F \) derivable from \( E' \) such that each fact in a derivation of \( F \) is contained in \( C^\infty \); thus, \( C \) identifies the part of \( J' \) to recompute. Rather than storing \( L \) directly, we adapt the \( r\)-materialisation algorithm by Motik et al. (2015a) and represent \( L \) by its rewriting \( (\gamma, P \setminus \hat{P}) \); the role of the two sets \( P \) and \( \hat{P} \) is discussed shortly. Lines 36–40 compute the facts in \( L \) derivable immediately from \( E' \); we iterate over each \( F \in C \) and each \( G \in F^{\infty} \); since we represent \( L \) by its rewriting, we add \( \gamma(G) \) to \( P \). The roles of set \( Y \) and lines 37–39 will be discussed shortly. Lines 41–50 compute the facts in \( L \) derivable using rules: we consider each fact \( F \in P \setminus \hat{P} \) (lines 41–42), each rule \( r \), and each match \( \sigma \) of \( F \) to a body atom of \( r \) (line 48), we evaluate the remaining body atoms of \( r \) (line 49), and we derive \( \gamma(h(r)\tau) \) for each match \( \tau \) (line 50). This basic idea is slightly more complicated by rewriting: if \( F = a \approx b \), we modify \( \gamma \) so that one constant becomes the representative of the other one (line 45). As a consequence, facts can become ‘outdated’ w.r.t. \( \gamma \), so we keep track of such facts using \( \hat{P} \): if \( F \) is ‘outdated’, we add \( F \) to \( \hat{P} \) and \( \gamma(F) \) to \( P \) (line 44); due to the latter, \( P \setminus \hat{P} \) eventually contains all ‘up to date’ facts. Finally, we apply the reflexivity rules \((\approx_1)\) to \( F \) (line 47).

Procedure saturate() is repeatedly called in \( B/F^\infty \). Set \( C \), however, never shrinks between successive calls, so set \( L \) never shrinks either; hence, at each call we can just continue the computation instead of starting ‘from scratch’. A minor problem arises if we derive a fact \( F \) with \( F \not\in C^\infty \) and so we do not add \( \gamma(F) \) to \( \hat{P} \), but \( C \) is later extended so that \( F \in C^\infty \) holds. We handle this by maintaining a set \( Y \) of ‘delayed’ facts: in line 59 we add \( F \) to \( Y \) if \( F \not\in C^\infty \); and in line 40 we identify each ‘delayed’ fact \( G \in C^\infty \cap Y \) and add \( \gamma(G) \) to \( P \). Procedure rewrite(a, b) implements rewriting: we update \( \gamma \) (line 52), apply the replacement rules \((\approx_1)\)–\((\approx_3)\) to already processed facts containing ‘outdated’ constants (line 54), ensuring that \( \Gamma \) is normal w.r.t. \( \gamma \) (line 56), and reapply the normalised rules (lines 57–58). Motik et al. (2015a) discuss in detail the issues related to rule updating and reevaluation.

**Procedure** checkProvability() takes a fact \( F \in I \) and ensures that, for each \( G \in F^\infty \), we have \( G \in J \) iff \( \gamma(G) \in P \setminus \hat{P} \)—that is, we know the correct status of each fact that \( F \) represents. To this end, we add \( F \) to \( C \) (line 22) and thus ensure that \( \gamma(P \setminus \hat{P}) \) correctly represents \( L \) (line 23). Each fact is added to \( \hat{C} \) only once, which guarantees termination of the recursion. We then use backward chaining to examine facts occurring in proofs of \( F \) and recursively check their provability: we stop at any point during that process if all facts in \( F^\infty \) become provable (lines 24, 28, 31, and 35). Lines 25–24 handle the reflexivity rules \((\approx_1)\): to check provability of \( c \approx c \), we recursively check the provability each fact containing \( c \). Lines 29–31 handle replacement rules \((\approx_1)\)--\((\approx_2)\): we recursively check the provability of \( c \approx c \) for each constant \( c \) occurring in \( F \). Finally, lines 32–35 handle the rules in \((\Pi)\): we consider each rule \( r \in \pi(\Pi) \) whose head matches \( F \) and each substitution \( \tau \) that matches the body of \( r \) in \( I \), and we recursively check the provability of \( b(r)\tau \).

**Procedure** \( BF^\infty() \) computes the set \( D \subseteq I \) of ‘doubtful’ facts. After initialising \( D \) to \( \pi(E^-) \) (lines 3–4), we consider each fact \( F \in D \) (lines 5–16) and determine whether some \( G \in F^\infty \) is no longer provable (line 6); if so, we add to \( D \) all facts that might be affected by the deletion of \( G \). Lines 9–11 handle rules \((\approx_1)\)--\((\approx_3)\); line 12 handles rules \((\approx_4)\); and lines 13–15 handle \((\Pi)\): we identify each rule \( r \in \pi(\Pi) \) where \( F \) matches a body atom of \( r \), we evaluate the remaining body atoms \( \pi \) of \( r \) in \( I \), and we add \( h(r)\tau \) to \( D \) for each \( \tau \) such that \( b(r)\tau \not\in I \). Once \( D \) is processed, \((\gamma, P \setminus \hat{P}) \) reflects the changes to \((\pi, I)\), which we exploit in Algorithm 2.

### 3.2 Optimisations

**Reflexivity.** Facts of the form \( F = c \approx c \) can be expensive for backward chaining: due to reflexivity rules \((\approx_1)\), in lines 25–28 we may end up recursively proving each fact \( G \) that mentions \( c \). However, \( F \) holds trivially if \( \gamma(F) \) contains a fact mentioning \( c \), in which case we can consider \( F \) proven and avoid any recursion. This is implemented in lines 37–39.

**Avoiding Redundant Derivations.** Assume that \( \Gamma \) contains a rule \( y_1 \equiv y_2 \leftarrow \langle x, R, y_1 \rangle \land \langle x, R, y_2 \rangle \), and consider a call to saturate() in which facts \( \langle a, R, b \rangle \) and \( \langle a, R, d \rangle \) both end up in \( P \). Unless we are careful, in line 50 we might consider substitution \( \tau_1 = \{ x \mapsto a, y_1 \mapsto b, y_2 \mapsto d \} \) twice: once when we match \( \langle a, R, b \rangle \) to \( \langle x, R, y_1 \rangle \), and once when we match \( \langle a, R, d \rangle \) to \( \langle x, R, y_2 \rangle \). Such redundant derivations can substantially degrade performance.

To solve this problem, set \( V \) keeps track of the processed subset of \( P \): after we extract a fact \( F \) from \( P \), in line 42 we transfer \( F \) to \( V \); moreover, in line 49 we evaluate rule bodies in \( V \setminus \hat{P} \) instead of \( P \setminus \hat{P} \). Now if \( \langle a, R, b \rangle \) is processed before \( \langle a, R, d \rangle \), at that point we have \( \langle a, R, d \rangle \not\in V \), so \( \tau_1 \) is not returned as a match in line 49; the situation when \( \langle a, R, d \rangle \) is processed first is analogous. This, however, does not eliminate all repetition: \( \tau_2 = \{ x \mapsto a, y_1 \mapsto b, y_2 \mapsto b \} \) is still considered when \( \langle a, R, b \rangle \) is matched to either of the two body atoms in the rule. Therefore, we annotate (see Section 3.3) the body atoms of rules so that, whenever \( F \) is matched to some body atom \( B_i \), no atom \( B_j \) preceding \( B_i \) in the body of \( r \) can be matched to \( F \). In our example, \( \tau_2 \) is thus
considered only when \(\langle a, R, b \rangle\) is matched to \(\langle x, R, y_t \rangle\).

B/F\(\approx\) avoids redundant derivations in similar vein: set \(O\) tracks the processed subset of \(D\); in lines 10 and 14 we match the relevant rules in \(I \setminus O\); and in line 16 we add a fact to \(O\) once it has been processed.

**Disproved Facts.** For each \(F \in I\) with \(F^n \cap J' = \emptyset\), no fact in \(F^n\) participates in a proof of any fact in \(J'\). Thus, in line 7 we collect all such facts in a set \(S\) of disproved facts, and in lines 26, 29, and 33 we exclude from \(\pi\) from backward chaining.

**Singletons.** If we encounter \(F = c \approx c\) in line 9 or 29 where \(c\) represents only itself (i.e., \(|c^n| = 1\)), then we know that no fact in \(F^n\) can derive a new fact using rules \((\approx_1)-(\approx_3)\), and so we can avoid considering rules \((\approx_1)-(\approx_3)\).

### 3.3 Formalisation

We borrow the notation by Motik et al. (2015b) to formalise B/F\(\approx\). We recapitulate some definitions, present the pseudo-code, and formally state the algorithm’s properties.

Given a dataset \(X\) and a fact \(F\), operation \(X.\text{add}(F)\) adds \(F\) to \(X\), and operation \(X.\text{delete}(F)\) removes \(F\) from \(X\); both return \(t\) if \(X\) was changed. For iteration, operation \(X.\text{next}\) returns the next fact from \(X\), or \(e\) if no such fact exists.

An annotated query has the form \(Q = B_1 \approx \cdots \approx B_k\), where each \(B_i\) is an atom and annotation \(\bowtie\) is either empty or equal to \(\neq\). Given datasets \(X\) and \(Y\) and a substitution \(\sigma\), operation \(X.\text{eval}(Q, Y, \sigma)\) returns a set containing each smallest substitution \(\tau\) such that \(\sigma \subseteq \tau\) and, for \(1 \leq i \leq k\), (i) \(B_i.\tau\) in \(X\) if \(\bowtie\) is empty or (ii) \(B_i.\tau\) in \(X\) if \(\bowtie\) is \(\neq\). We often write \([Z \setminus W]\) instead of \(X\), meaning that \(Q\) is evaluated in the difference of sets \(Z\) and \(W\).

Given a fact \(F\), operation II.\text{matchHead}(F) returns all tuples \(\langle r, Q, \sigma\rangle\) with \(r \in \Pi\) a rule of the form (1), \(\sigma\) a substitution such that \(H.\sigma = F\), and \(Q = B_1 \approx \cdots \approx B_n\). Moreover, operation II.\text{matchBody}(F) returns all tuples \(\langle r, Q, \sigma\rangle\) with \(r \in \Pi\) a rule of the form (1), \(\sigma\) a substitution such that \(B_i.\sigma = F\) for some \(1 \leq i \leq n\), and \(Q\) is defined as

\[
Q = B_1 \approx \cdots \approx B_{i-1} \approx B_{i+1} \approx \cdots \approx B_n. \tag{2}
\]

Finally, given a mapping \(\gamma\) of constants to constants, and constants \(d\) and \(c\), operation \(\gamma.\text{mergeLto}(d, c)\) modifies \(\gamma\) so that \(\gamma(e) = c\) holds for each constant \(e\) with \(\gamma(e) = d\).

B/F\(\approx\) consists of Algorithms 1–7. Theorem 1 shows that the algorithm is correct and that, just like the seminal algorithm [Abiteboul et al., 1995], it does not repeat derivations; the proof is given in the appendix.

### 4 Evaluation

We have implemented and evaluated the B/F\(\approx\) algorithm in the open-source RDF data management system RDFox. The system and the test data are all available online.\(^2\)

**Objectives.** Updates can be handled either incrementally or by rematerialisation, and equality can be handled either by rewriting or by axiomatisation, giving rise to four possible approaches to updates. Our first objective was to compare all of them to determine their relative strengths and weaknesses.

As \(E^-\) increases in size, incremental update becomes harder, but rematerialisation becomes easier. Thus, our second objective was to investigate the relationship between the update size and the performance of the respective approaches.

**Datasets.** Equality is often used in OWL ontologies on the Semantic Web, so we based our evaluation on several well-known synthetic and ‘real’ RDF datasets.

Each dataset comprises an OWL ontology and a set of explicit facts. UOBM [Ma et al., 2006] extends LUBM [Guo et al., 2005], and we used the data generated for 100 universities; we did not use LUBM because it does not use \(\approx\). Claros contains information about cultural artefacts.\(^3\) DBpedia consists of structured information extracted from Wikipedia.\(^4\) UniProt is a knowledge base about protein sequences;\(^5\) we selected a subset of the original (very large) set of facts. Finally, OpenCyc is an extensive, manually curated upper ontology.\(^6\)

Following Zhou et al. (2013), we converted the ontologies into lower (L) and upper bound (U) programs: the former is the OWL 2 RL subset of the ontology transformed into datalog as described by Grosof et al. (2003), and the latter captures all consequences of the ontology using an unsound approximation. Upper bound programs are interesting as they tend to be ‘hard’. We also manually extended the lower bound (LE) of Claros with ‘hard’ rules (e.g., we defined related documents as pairs of documents that refer to the same topic).

**Update Sets.** For each dataset, we randomly selected several subsets \(E^-\) of \(E\). We considered small updates of 100 and 5k facts on all datasets. Moreover, for each dataset we identified the ‘equilibrium’ point \(n\) at which B/F\(\approx\) and Remat\(\approx\) take roughly the same time. If \(n\) was large, we generated subsets \(E^-\) with sizes equal to 25%, 50%, 75%, and 100% of \(n\); otherwise, we divided \(n\) in an ad hoc way.

**Test Setting.** We used a Dell server with two 2.60GHz Intel Xeon E5-2670 CPUs and 256 GB of RAM running Fedora release 20, kernel version 3.17.7-200.fc20.x86_64.

**Test Results.** Table 1 summarises our test results. For each dataset, we show the numbers of explicit facts \(|E|\) and rules \(|\Pi|\), the number of facts in the initial r-materialisation \(|I^n|\), and the time \((T)\) and the number of derivations \((D)\) used to compute it; rewriting; moreover, we show the latter three numbers for the initial materialisation computed using axiomatised equality \(|I^A|\), \(|T^A|\), and \(|D^A|\). For each set \(E^-\), we show the numbers \(|E^-|\) and \(|D^-|\) of deleted facts with rewriting and axiomatisation, respectively, as well as the times \((T)\) and the number of derivations \((D)\) for each of the four update approaches. All times are in seconds. We could not complete all axiomatisation tests with Claros-LE as each run took about two hours. Due to the upper bound transformation, the r-materialisation of UOBM-100-U contains a constant \(c\) with \(|c^n| = 3930\); thus, when \(n \approx\) is axiomatised, deriving just all equalities involving \(c^n\) requires 3930\(^3\) = 60 billion derivations, which causes the initial materialisation to last longer than four hours. The number of derivations \(D\) in

\(\footnotesize{\text{\url{http://www.clarosnet.org/XDB/ASP/clarosHome/}}}\)
\(\footnotesize{\text{\url{http://dbpedia.org/}}}\)
\(\footnotesize{\text{\url{http://www.uniprot.org}}}\)
\(\footnotesize{\text{\url{http://www.cyc.com/platform/openyc}}}\)
Input Variables

- $E$ : the explicit facts
- $\Pi$ : the datalog program
- $(\pi, I)$ : the r-materialisation of $E$ w.r.t. $\Pi$
- $E^-$ : the facts to delete from $E$

Global Temporary Variables

- $D$ : the consequences of $E^-$ that might require deletion
- $O$ : the processed subset of $D$
- $C$ : the facts whose provability must be checked
- $\gamma$ : the mapping recording the changes needed to $\pi$
- $P$ : the proved facts
- $\hat{P}$ : the proved rewritten facts
- $Y$ : the proved facts not in $C_{\pi}$
- $V$ : the processed subset of $P$
- $S$ : the set of disproved facts

Algorithm 1 B/$F^\pi$()

2. initialise $\gamma$ as identity and $\Gamma := \Pi$
3. for each $F \in E^-$ do
4. if $E$.delete($F$) then $D$.add($\pi(F)$)
5. while $(F := D$.next$) \neq \varepsilon$ do
6. checkProvability($F$)
7. for each $G \in C$ s.t. allDisproved($G$) do $S$.add($G$)
8. if not allProved($F$) then
9. if $F = c \equiv c$ and $|c^\pi| > 1$ then
10. for each $G \in I \setminus O$ with $c \in \text{voc}(G)$ do
11. $D$.add($G$)
12. for each $c \in \text{voc}(F)$ do $D$.add($c \equiv c$)
13. for each $(r, Q, \sigma) \in \pi(\Pi), \text{matchBody}(F)$ do
14. for each $\tau \in [I \setminus O].\text{eval}(Q, \{F\}, \sigma)$ do
15. $D$.add($h(\tau)$)
16. $O$.add($F$)
17. propagateChanges()

Algorithm 2 propagateChanges()

18. for each $c \equiv c \in C$ and each $d$ with $\pi(d) = c$ do
19. $\pi(d) := \gamma(d)$
20. for each $F \in D \setminus (P \setminus \hat{P})$ do $I$.delete($F$)
21. for each $F \in P \setminus \hat{P}$ do $I$.add($\pi(F)$)

Algorithm 3 Auxiliary functions

allProved($F$):

- if $F \notin S$ and $\gamma(F^\pi) \subseteq (P \setminus \hat{P})$ allDisproved($F$):
- if $\gamma(F^\pi) \cap (P \setminus \hat{P}) = \emptyset$

Theorem 1. Let $(\pi, I)$ be the r-materialisation of a dataset $E$ w.r.t. a program $\Pi$, and let $E^-$ be a dataset.

1. Algorithm 1 terminates, at which point $(\pi, I)$ contains the r-materialisation of $E \setminus E^-$ w.r.t. $\Pi$
2. Each combination of a rule $r$ and a substitution $\tau$ is considered at most once in line 50 or line 58, but not both.
3. Each combination of a rule $r$ and a substitution $\tau$ is considered at most once in line 15.
B/F\(\infty\) is the sum of the number of times a fact is determined as ‘doubtful’ (lines 11, 12, and 15), checked in backward chaining (lines 27, 30, and 34), or derived in forward chaining (line 59); we use this number to estimate reasoning difficulty independently from implementation details.

**Discussion.** For updates of 100 facts, B/F\(\infty\) outperforms all other approaches, often by orders of magnitude, and in most cases it does so even for much larger updates.

Even when \(|A| - |I|\) is ‘small’ (i.e., when not many equalities are derived), B/F\(\infty\) outperforms B/F\(A\). This seems to be mainly because B/F\(A\) ascribes no special meaning to \(I_0\), and so it does not use the optimisation from lines 37–39, thus, when trying to prove \(c \approx e\), B/F\(A\) performs backward chaining via rules \((\approx)\) and so it potentially examines each fact containing \(c\). On Claros-L, although \(|A|\) and \(|I|\) are of similar sizes, \(I^C\) contains one constant \(c\) with \(|c^C| = 306\), which gives rise to 306\(^3\) derivations; this explains the difference in the performance of B/F\(\infty\) and B/F\(A\).

Remat\(\infty\) outperforms B/F\(\infty\) in cases similar to those described by Motik et al. (2015b). For example, in UOBM, relation hasSameHomeTownWith is symmetric and transitive, which creates cliques of connected constants; B/F always re-computes each changed clique, thus repeating most of the ‘hard’ work. Equality connects constants in cliques, which poses similar problems for B/F\(\infty\). For example, due to the constant \(c\) with \(|c^C| = 3930\), deleting 5k facts in UOBM-100-U results in only 961k (about 1.2\% of \(|I^C|\)) facts being added to set \(C\) in line 22, but these facts contribute to 73\% of the derivations from the initial r-materialisation; thus, B/F\(\infty\) repeats in Algorithm 5 a substantial portion of the initial work.

On OpenCyc-L, Remat\(\infty\) already outperforms B/F\(\infty\) on updates of 1k triples, which was surprising since the former makes more derivations than the latter. Our investigation revealed that OpenCyc-L contains about 200 rules of the form \((x, y) \leftarrow (x, R_i, y)\) that never fire during forward chaining; however, to check provability of \((a, \text{type}, C)\), Algorithm 4 considers in line 32 each time each of the 200 rules. After removing all such ‘idle’ rules manually, B/F\(\infty\) and Remat\(\infty\) could update 1k tuples in roughly the same time. Further analysis revealed that the slowdown in B/F\(\infty\) occurs mainly in line 40: the condition is checked for 13.3M facts \(F\), and these give rise to 139M facts in \(F^C\), each requiring an index lookup; the latter number is similar to the number of derivations in rematerialisation, which explains the slowdown. We believe one can check this condition more efficiently by using additional book-keeping.

<table>
<thead>
<tr>
<th>B/F(\infty)</th>
<th>Remat(\infty)</th>
</tr>
</thead>
</table>

**Table 1: Experimental results**
5 Conclusion

This paper describes what we believe to be the first approach to incremental maintenance of datalog materialisation when the latter is computed using rewriting—a common optimisation used when programs contain equality. Our algorithm proved to be very effective, particularly on small updates.

In our future work, we shall aim to address the issues we identified in Section 4. For example, to optimise the check in line 40, we shall investigate ways of keeping track of how explicit facts are merged so that we can implement the test by iterating over the appropriate subset of $E$ rather than over $F^*$. Moreover, we believe we can considerably improve the efficiency of both the initial materialisation and the incremental updates by using specialised algorithms for rules that produce large cliques; hence, we shall identify common classes of ‘hard’ rules and then develop such specialised algorithms.

Acknowledgments

This work was funded by the EPSRC projects MaSI, Score!, and DBOnto, and the FP7 project Optique.

References


A Proof of Theorem 1

Let \( \Pi \) be a program (that ascribes no special meaning to \( \approx \)), and let \( E \) be a dataset. A derivation tree for a fact \( F \) w.r.t. \( \Pi \) is a finite tree \( T \) in which each node \( t \) is labelled with a fact \( F_t \) and each nonleaf node \( t \) is labelled with a rule \( r_t \in \Pi \) and a substitution \( \sigma_t \) such that the following holds:

- **D1.** \( F_e = F \) holds for the root \( e \) of \( T \);
- **D2.** \( F_t \in E \) holds for each leaf node \( t \) of \( T \); and
- **D3.** \( h(r_t)\sigma_t = F_t \) and \( b(r_t)\sigma_t = \{F_{t_1}, \ldots, F_{t_n}\} \) hold for each nonleaf node \( t \) of \( T \) with children \( t_1, \ldots, t_n \).

The materialisation \( \Pi^\infty(E) \) of \( E \) w.r.t. \( \Pi \) is the smallest set containing each fact that has a derivation tree from \( E \) w.r.t. \( \Pi \); this definition of \( \Pi^\infty(E) \) is equivalent to the one in Section 2. The height of a derivation tree is the length of its longest branch; moreover, the height of a fact \( F \in \Pi^\infty(E) \) w.r.t. \( E \) and \( \Pi \) is the minimum height of a derivation tree for \( F \) from \( E \) w.r.t. \( \Pi \).

In the rest of this paper, we make the following assumption (*): no derivation tree contains a node \( t \) where \( r_t \) is \((\approx_1)\) and \( \sigma_t(x_1) = \sigma_t(x'_1) \), or \( r_t \) is \((\approx_2)\) and \( \sigma_t(x_2) = \sigma_t(x'_2) \), or \( r_t \) is \((\approx_3)\) and \( \sigma_t(x_3) = \sigma_t(x'_3) \). This is w.l.o.g. because, for each such \( t \), we have \( F_t = F_{t_1} \) for \( t_1 \) the first child of \( t \): hence, we can always remove such \( t \) from the derivation tree.

Next, we recapitulate Theorem 1 and present its proof, which we split into several claims.

**Theorem 1.** Let \( (\pi, I) \) be the r-materialisation of a dataset \( E \) w.r.t. a program \( \Pi \), and let \( E' \) be a dataset.

1. Algorithm 1 terminates, at which point \( (\pi, I) \) contains the r-materialisation of \( E \setminus E' \) w.r.t. \( \Pi \).
2. Each combination of a rule \( r \) and a substitution \( \tau \) is considered at most once in line 50 or line 58, but not both.
3. Each combination of a rule \( r \) and a substitution \( \tau \) is considered at most once in line 15.

In the rest of this section, we fix a datalog program \( \Pi \) and datasets \( E \) and \( E' \). Let \( (\pi, I) \) be the r-materialisation of \( E \) w.r.t. \( \Pi \); let \( J := (\Pi \cup \Pi_{\approx})^\infty(E) \); let \( E' := E \setminus E' \); let \( (\pi', I') \) be the r-materialisation of \( E' \) w.r.t. \( \Pi \); and let \( J' := (\Pi \cup \Pi_{\approx})^\infty(E') \).

By the monotonicity of datalog, we clearly have \( J' \subseteq J \).

We next show that Algorithm 5 essentially captures the r-materialisation algorithm by Motik et al. (2015a).

**Claim 1.** Let \( P \) and \( \hat{P} \) be as obtained after a call to Algorithm 5 in line 23, let \( K := \{d = d' \mid d \in \text{voc}(E)\} \), and let \( L \) be the set containing precisely each fact \( F \) that has a derivation tree from \( K \cup E' \) w.r.t. \( \Pi \cup \Pi_{\approx} \) in which \( F_t \in C^\pi \) holds for each node \( t \) of \( T \). Then, the following properties hold:

1. \( \gamma(c) = \min E_c(L) \) for each constant \( c \);
2. \( P \setminus \hat{P} = \gamma(L) \); and
3. each combination of a rule \( r \) and a substitution \( \tau \) is considered at most once in line 50 or line 58, but not both.

**Proof (Sketch).** Algorithm 5 is a variant of the r-materialisation algorithm by Motik et al. (2015a), so properties 1–3 hold by a straightforward modification of the correctness proof of that algorithm. This proof is quite lengthy so, for the sake of brevity, we just summarise the differences.

- Lines 37–39 ensure \( \gamma(C^\pi \cap K) \subseteq P \setminus \hat{P} \), and line 40 ensures \( \gamma(C^\pi \cap E') \subseteq P \setminus \hat{P} \); hence, \( C^\pi \cap (K \cup E') \) plays the same role that explicit facts play in the algorithm by Motik et al. (2015a).
- Let \( F \) be an arbitrary fact considered in line 41. To ensure property 4 of Claim 1, the algorithm by Motik et al. (2015a) uses slightly different annotated queries to apply the rules in lines 48–49 only to facts extracted before \( F \). In contrast, Algorithm 7 keeps track of previously processed facts in set \( V \), but this has exactly the same effect.
- All derivations of a fact in line 47, 50, or 58, are handled by Algorithm 7, which, for each \( F \), checks whether \( \pi(F) \in C \); this is equivalent to checking \( F \in C^\pi \). If the latter holds, then \( F \) is added to \( P \), and otherwise \( F \) is added to \( Y \). If in a subsequent invocation of Algorithm 5 set \( C \) is extended such that \( \pi(F) \in C \) suddenly holds, then \( \gamma(F) \) is added to \( P \) in line 40. This, however, does not change the algorithm in any substantial way.

The following claim follows immediately from the definitions in Algorithm 3.

**Claim 2.** The following properties hold for an arbitrary fact \( F \) normal w.r.t. \( \pi \):

1. \( \text{allProved}(F) = t \) if and only if \( F \not\in S \) and \( F^\pi \subseteq (P \setminus \hat{P})^\gamma \); and
2. \( \text{allDisproved}(F) = t \) if and only if \( F^\pi \cap (P \setminus \hat{P})^\gamma = \emptyset \).

We next show that sets \( C, P, \hat{P}, S, \) and \( \gamma \) always satisfy an important property.

**Claim 3.** Assume that Algorithm 4 is applied to some fact \( F \), mapping \( \gamma \), and sets \( C, P, \) and \( \hat{P} \) where \( S \) is normal w.r.t. \( \pi \) and \( S^\pi \cap J' = \emptyset \), and assume that all of these satisfy the following property:

\( \langle \delta \rangle \) for each \( G \in C, \) either \( G^\pi \subseteq (P \setminus \hat{P})^\gamma \) or, for each fact \( H \in G^\pi \), each derivation tree \( T \) for \( H \) from \( E' \) w.r.t. \( \Pi \cup \Pi_{\approx} \), and each child \( t_i \) of the root \( t \) of \( T \), we have \( \pi(F_{t_i}) \in C \).
Then, property \( (\Diamond) \) remains preserved after the invocation of Algorithm 4.

**Proof.** The proof is by induction on recursion depth of Algorithm 4 at which a fact is added to \( C \). For the induction base, \( (\Diamond) \) remains preserved if the algorithm returns in line 22.

For the induction step, assume that \( (\Diamond) \) holds for each fact \( G \in C \) different from \( F \) after a recursive call in line 27, 30, or 34. If the algorithm returns in line 24, 28, 31, or 35, then property 1 of Claim 2 implies \( F^\pi \subseteq (P \setminus \hat{P})^\gamma \), so property \( (\Diamond) \) remains preserved. Otherwise, consider an arbitrary fact \( H \in F^\pi \) and an arbitrary derivation tree \( T \) for \( H \) from \( E' \) w.r.t. \( \Pi \cup \Pi_\infty \).

Let \( t_1, \ldots, t_n \) be the children (if any exist) of the root \( \epsilon \) of \( T \); since \( J \) contains each fact labelling a node of \( T \), we have \( \{F_1, \ldots, F_n\} \subseteq J' \subseteq J \). Now let \( F_i = \pi(F_{t_i}) \); by the definition of r-materialisation, we have \( \{F_1, \ldots, F_n\} \subseteq I \). Moreover, for each \( 1 \leq i \leq n \), we have \( F_i \in J' \) and \( S^\pi \cap J' = \emptyset \), which imply \( F_i \notin S^\pi \); moreover, \( S \) is normal w.r.t. \( \pi \), so \( F_i \notin S \) as well. Finally, we clearly have \( \pi(r_{t_i}c) = (\pi(r_{t_i})\pi(c)) = F \) and \( \pi(S) = \pi((F_1, \ldots, F_n) \subseteq I \setminus S \).

We next consider the forms of \( r_{t_i}c \).

- Assume \( r_{t_i}c \) is of the form \( (\approx^\pi) \), so \( n = 1 \). Fact \( F_1 \) is eventually considered in line 26, so, due to the recursive call in line 27, we have \( F_1 \in C \), as required.

- Assume \( r_{t_i}c \) is of the form \( (\approx^\pi) \); thus, \( n = 2 \), \( F_1 = F \), and \( F_2 = c \approx c \) for some constant \( c \). Fact \( F_1 = F \) is added to \( C \) in line 22. Moreover, by assumption \( (\ast) \) on the shape of \( T \), we have \( F_2 = s \approx t \) with \( s \neq t \); since \( (s) = (t) = c \), we have \( |c^\pi| > 1 \). Thus, due to the recursive call in line 30, we have \( F_2 \in C \), as required.

- Assume \( r_{t_i}c \in \Pi \). Then, \( \pi(r_{t_i}) \in \pi(\Pi) \), so \( \pi(r_{t_i}) \) and \( \pi(c) \) are eventually considered in lines 32 and 33; hence, due to the recursive call in line 34, we have \( F_i \in C \) for each \( 1 \leq i \leq n \), as required.

Calls in line 6 ensure another property on \( C, P, \hat{P}, \) and \( S \).

**Claim 4.** The following properties hold after each line of Algorithm 1:

1. property \((\Diamond)\) is satisfied;
2. \( (P \setminus \hat{P})^\gamma = C^\pi \cap J' \);
3. \( \gamma(c) = \min E_c(C^\pi \cap J') \) for each constant \( c \); and
4. \( S^\pi \cap J' = \emptyset \).
5. For each fact \( F \in O \), we have \( F^\pi \not\subseteq J' \).
6. \( D \subseteq C \).

**Proof.** The proof is by induction on the number of iterations of the loop in lines 5–16. For the induction base, we have \( S = C = P = O = \emptyset \) in line 1, so properties 1–5 clearly hold initially. For the induction step, assume that all properties hold before line 6. Due to property 4 and Claim 3, property 1 remains preserved after line 6; hence, we next consider properties 2–6.

(Property 2) Let \( K \) and \( L \) be as stated in Claim 1; note that property 2 of Claim 1 is equivalent to \((P \setminus \hat{P})^\gamma = L \). We first show \((P \setminus \hat{P})^\gamma \subseteq C^\pi \cap J' \). Since \( K \subseteq J', \) we clearly have \( J' = (\Pi \cup \Pi_\infty)^\infty(K \cup E') \). Moreover, for each \( F \in (P \setminus \hat{P})^\gamma \) we have \( F \in L \), so by the definition of \( L \) there exists a derivation tree \( T \) for \( F \) from \( K \cup E' \) w.r.t. \( P \cup \Pi_\infty \) such that \( F_t \in C^\pi \) holds for each node \( t \) of \( T \); but then, we clearly have \( F \in C^\pi \cap J' \). We next prove \( C^\pi \cap J' \subseteq (P \setminus \hat{P})^\gamma \) by induction on the height \( h \) of a fact \( F \in C^\pi \cap J' \) w.r.t. \( E' \) and \( \Pi \cup \Pi_\infty \).

- If \( h = 0 \), then \( F \in E' \); since \( F \in C^\pi \), by the definition of \( L \) we have \( F \in L \); but then, \( F \in (P \setminus \hat{P})^\gamma \) as well.

- Assume that the claim holds for each fact \( F \in C^\pi \cap J' \) whose height w.r.t. \( E' \) and \( \Pi \cup \Pi_\infty \) is at most \( h \), and consider an arbitrary fact \( F \in C^\pi \cap J' \) with height \( h + 1 \); let \( T \) be the corresponding derivation tree for \( F \). Moreover, assume that \( F \notin (P \setminus \hat{P})^\gamma \); then, \( F \in C^\pi \) implies \( \pi(F) \in C \); hence, property \((\Diamond)\) ensures that, for each child \( t_i \) of the root of \( T \), we have \( \pi(F_{t_i}) \in C^\pi \), which is equivalent to \( F_{t_i} \in C^\pi \). Now the height of each \( F_{t_i} \) w.r.t. \( E' \) and \( \Pi \cup \Pi_\infty \) is at most \( h \) so, by the induction assumption, we have \( F_{t_i} \in (P \setminus \hat{P})^\gamma = L \). The latter ensures that, for each \( F_{t_i} \), there exists a derivation tree \( T_{t_i} \) in which each node is labelled by a fact contained in \( C^\pi \). Let \( T' \) be the derivation tree in which the root \( \epsilon \) is labelled with the same fact, rule, and substitution as in \( T \), and each \( T_i \) is a subtree of \( \epsilon \). Clearly, \( T' \) is a derivation tree for \( F \) from \( E' \) w.r.t. \( \Pi \cup \Pi_\infty \) in which each node is labelled by a fact contained in \( C^\pi \); thus, by the definition of \( L \), we have \( F \in L = (P \setminus \hat{P})^\gamma \), as required.

(Property 3) This property follows directly from property 1 of Claim 1 and property 2 of Claim 4.

(Property 4) Assume that some fact \( G \) is added to \( S \) in line 7. Then allDisproved\((G) = t \), which by property 2 of Claim 2 implies \( G^\pi \cap (P \setminus \hat{P})^\gamma = \emptyset \). Property 2 of Claim 4 holds at this point, so we have \( G^\pi \subseteq C^\pi \cap J' = \emptyset \). Finally, lines 6 and 22 ensure \( G \in C \), so we have \( G^\pi \subseteq C^\pi \); thus, \( G^\pi \cap J' = \emptyset \), and so adding \( G \) to \( S \) preserves property 4.

(Property 5) Assume that some fact \( F \) is added to \( O \) in line 16. Then allProved\((F) = t \), which by property 1 of Claim 2 implies \( F \in S \) or \( F^\pi \not\subseteq (P \setminus \hat{P})^\gamma \). In the former case, \( F^\pi \not\subseteq J' \) holds directly from property 4. In the latter case, property 2 of
Claim 4 holds at this point, so we have $F^\pi \not\subseteq C^\pi \cap J'$; moreover, lines 6 and 22 ensure $F \in C$, which implies $F^\pi \subseteq C^\pi$; this, in turn, implies $F^\pi \not\subseteq J'$. Consequently, adding $F$ to $O$ preserves property 5.

(Property 6) Each fact $F$ extracted from $D$ in line 5 is passed in line 6 to Algorithm 4, which in turn ensures that $F$ is added to $C$ in line 22. 

We next show that set $D$ contains each fact that needs to be deleted, and each fact that contains a constant whose representative changes as a result of the update.

**Claim 5.** For each fact $F \in J \setminus J'$, the following two properties hold in line 17:

1. $F(\pi) \in D$, and
2. if $F = s \approx t$ with $s \neq t$, then $D$ contains each fact $G \in I$ such that $\pi(s) \in \text{voc}(G)$ and $G^\pi \not\subseteq J'$.

**Proof.** Consider an arbitrary fact $F \in J \setminus J'$.

(Property 1) We prove the claim by induction on the height $h$ of $F$ w.r.t. $E$ and $\Pi \cup \Pi_\approx$; the notion of the height of $F$ is correctly defined because $F \in J$. For the induction base, assume $h = 0$; now $F \in J$ implies $F \in E$; moreover, $F \not\subseteq J'$ implies $F \not\subseteq E'$; thus, $F \in E'$, and so $\pi(F)$ is added to $D$ in lines 3–4. For the induction step, assume that the claim holds for each fact in $J \setminus J'$ whose height w.r.t. $E$ and $\Pi \cup \Pi_\approx$ is at most $h$, and assume that the height of $F$ w.r.t. $E$ and $\Pi \cup \Pi_\approx$ is $h + 1$. Let $T$ be a corresponding derivation tree for $F$ from $E$ w.r.t. $\Pi \cup \Pi_\approx$; let $t_1, \ldots, t_n$ be the children of the root of $T$; and let $F_i = \pi(F_{t_i})$ for each $1 \leq i \leq n$. Moreover, let $N$ contain precisely each $F_i$, $1 \leq i \leq n$, such that $F_i \in D$ and $F_i^\pi \not\subseteq J$. Since $F \not\subseteq J'$, some $j$ with $1 \leq j \leq n$ exists such that $F_j \not\subseteq J'$; moreover, $T$ is a derivation tree for $F$ from $E$ w.r.t. $\Pi \cup \Pi_\approx$, so $F_j \in J$ and the height of $F_j$ is at most $h$; but then, we have $\pi(F_j) 

• Assume that $r_i$ is $(\approx_1)$–$(\approx_3)$. Then, we clearly have $\pi(F) = F_1$; fact $F_1$ is of the form $F_1 = s \approx t$ with $s \neq t$ and $s \in C$; we have two possible ways to choose $F'$. If $F' = F_1$, then $F(\pi) = F_1 = F^\pi \in D$ holds. If $F' = F_2$, then $s \neq t$ by assumption $(s)$ on the shape of $T$, so $|e^\pi| > 1$ and the check in line 9 passes; furthermore, due to $F_1 \in I \setminus O$, we eventually consider fact $G = F_1 = \pi(F)$ in line 10 and add it to $D$ in line 11.

• Assume that $r_i$ is $(\approx_4)$. Then, $F$ is of the form $s \approx s$ so $\pi(F) = c \approx c$ for $c = \pi(s)$; clearly, we have $c \in \text{voc}(F')$ and $F' = F_1$. But then, $\pi(F)$ is added to $D$ in line 12.

• Assume that $r_i \in \Pi$. We clearly have $\pi(r_i, \sigma) = \pi(r_i) \pi(\sigma)$; therefore, we have $\pi(F) = \pi(h(r_i, \sigma)) = h(\pi(r_i)) \pi(\sigma)$ and $\pi(b(r_i, \sigma)) = \{F_1, \ldots, F_n\} = b(\pi(r_i)) \pi(\sigma) \subseteq I \setminus O$. Moreover, we clearly have $\pi(r_i) \in \Pi$. Finally, let $i$ be the smallest integer with $1 \leq i \leq n$ such that $F_i = F'$, and let $Q$ be annotated query (2) obtained from $\pi(r_i)$ for that $i$; clearly, the way in which we chose $i$ ensures $F_i \neq F'$ for each $j$ with $1 \leq j < i$. All of these observations ensure together that $\pi(r_i, Q, \sigma)) \in \pi(\Pi), \text{matchBody}(F')$ is considered in line 13, and that $\pi(\sigma)$ is considered in line 14; consequently, $\pi(F)$ is added to $D$ in line 15.

(Property 2) Assume that $F$ is of the form $F = s \approx t$ with $s \neq t$, let $c = \pi(s) = \pi(t)$, and let $F' = \pi(F)$. Property 1 of this claim ensures $F' = c \approx c \in D \subseteq C$, and so we have $(F')^\pi \subseteq C^\pi$; but then, together with $F \not\subseteq J'$, property 2 of Claim 4 ensures $(F')^\pi \not\subseteq (P \setminus \hat{P})^\pi$; finally, property 1 of Claim 2 ensures all$\text{Proved}(F') = f$. Fact $F'$ is eventually processed in line 5, and by the previous discussion the check in line 8 passes. Moreover, $s \neq t$ implies $|e^\pi| > 1$, so the check in line 9 passes as well. Now consider an arbitrary fact $G \in I$ such that $c = \text{voc}(G)$ and $G^\pi \not\subseteq J'$; property 5 of Claim 4 ensures $G \not\subseteq O$, and therefore $G$ is added to $D$ in line 11.

We next show that Algorithm 1 correctly updates $I$ to $I'$. 

**Claim 6.** Algorithm 1 updates set $I$ to $I'$.

**Proof.** Property 6 of Claim 4 and property 1 of Claim 5 clearly ensure that (3) holds. Furthermore, property 2 of Claim 4 clearly ensures that (4) holds.

$$J \setminus J' \subseteq D^\pi \subseteq C^\pi$$  

$$\{P \setminus \hat{P}\}^\pi \subseteq J' \subseteq J$$

(3)  

(4)

For convenience we recapitulate the definitions of $\pi(e), \pi'(e)$, and $\gamma(e)$; note that (7) follows immediately from properties 2 and 3 of Claim 4. Finally, (4), (6), and (7) clearly imply (8).

$$\pi(e) = \min E_{\pi}(J)$$

(5)
\[ \pi'(c) = \min E_c(J') \quad (6) \]
\[ \gamma(c) = \min E_c((P \setminus \hat{P})') \quad (7) \]
\[ \pi'(P \setminus \hat{P} \gamma) = \pi'(P \setminus \hat{P}) \quad (8) \]

Before proceeding, we prove several useful properties. Consider an arbitrary constant \( c \) with \( \pi(c) = c \); by (4) and (5)–(7), we clearly have \( \pi'(c) = c \) and \( \gamma(c) = c \). Thus, for each fact \( F \) with \( \pi(F) = F \), we have \( \pi'(F) = F \) and \( \gamma(F) = F \), which ensures the following properties:

\[
\begin{align*}
F \in I & \iff F \in J, \\
F \in D & \iff F \in D^\pi, \quad \text{and} F \in C & \text{iff } F \in C^\pi.
\end{align*}
\]

We next show that lines 18–19 update \( \pi \) to \( \pi' \). To this end, consider arbitrary constants \( c \) and \( d \) with \( \pi(d) = c \), and let \( F = c \approx c \). Set \( F^\pi \) as defined above. Then, we have \( \pi(F) = F \) and \( \pi'(F) = F \), which implies \( F^\pi \approx F^\pi \). But then, property 2 of Claim 4 ensures \( F^\pi \in F' \). Due to the latter, some \( F \) is in \( F' \), and hence show that line 20 correctly deletes the relevant facts; please remember that, due to updates in lines 18–19, mapping \( \pi \) actually contains \( \pi' \) in line 21.

We next prove \( I \setminus I' = D \setminus (P \setminus \hat{P}) \) and hence show that line 21 correctly adds the relevant facts; please remember that, due to updates in lines 18–19, mapping \( \pi \) actually contains \( \pi' \) in line 21.

We finally prove that \( I' = [I \setminus (I \setminus I')] \cup \pi'(P \setminus \hat{P}) \) and hence show that line 21 correctly adds the relevant facts; please remember that, due to updates in lines 18–19, mapping \( \pi \) actually contains \( \pi' \) in line 21.

Claim 7. Each combination of a rule \( r \) and a substitution \( \tau \) is considered at most once in line 15.

\[ \begin{align*}
\pi'(c) &= \min E_c(J') \\
\gamma(c) &= \min E_c((P \setminus \hat{P})') \\
\pi'(P \setminus \hat{P}) &= \pi'(P \setminus \hat{P})
\end{align*} \]

\[ (6) \] (7) (8)