Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

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 $\Pi^\infty(E)$ of consequences of a set of *explicit* facts E and a datalog program Π is a key service in datalog systems. It can be supported by precomputing and storing $\Pi^\infty(E)$ so that queries can be evaluated directly, without further reference to Π. Set $\Pi^\infty(E)$ and the process of computing it are called the *materialisation* of E w.r.t. Π. This technique is used in the state of the art systems such as Olwgres [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 identifies 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.*, 2015b].

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 $\Pi^\infty(E)$ of E w.r.t. Π and a set of facts E^-, the main challenge for an incremental algorithm is to efficiently compute $\Pi^\infty(E \setminus E^-)$. 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 $F \in \Pi^\infty(E)$ 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 [Dewan *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 $\Pi^\infty(E)$ 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.*, 2015c].

Combining rewriting and incremental maintenance is difficult due to complex interactions between the two techniques: 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 axiomatise 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. Π computed using the rewriting approach by Motik *et al.* (2015b). 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[1] 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 $\langle t_1, t_2, t_3 \rangle$, 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; $\mathsf{h}(r) := H$ is the *head atom* of r; each B_i is a *body atom* of r; and $\mathsf{b}(r)$ is the set of all body atoms of r. A *(datalog) program* is a finite set of rules.

$$H \leftarrow B_1 \wedge \cdots \wedge B_n \tag{1}$$

A *substitution* is a partial mapping of variables to terms. For α a term, atom, rule, or a set of these, $\mathsf{voc}(\alpha)$ is the set of all constants in α, and $\alpha\sigma$ is the result of applying a substitution σ to α. The *materialisation* $\Pi^\infty(E)$ of a dataset E w.r.t. a program Π is the smallest superset of E containing $\mathsf{h}(r)\sigma$ for each rule $r \in \Pi$ and substitution σ with $\mathsf{b}(r)\sigma \subseteq \Pi^\infty(E)$.

Equality. The constant *owl:sameAs* (abbreviated $\approx$) can be used to encode equality between constants. For example, fact $\langle \mathsf{P.\_Smith}, \approx, \mathsf{Peter\_Smith} \rangle$ states that P._Smith and Peter_Smith are one and the same object. Facts of the form $\langle s, \approx, t \rangle$ are called *equalities* and, for readability, are abbreviated as $s \approx t$; note that $\approx\, \in \mathsf{voc}(s \approx t)$. Program $\Pi_\approx$ consisting of rules ($\approx_1$)–($\approx_4$) axiomatises $\approx$ as a congruence relation. If a program Π or a dataset E contain $\approx$, systems then answer queries in the materialisation of E w.r.t. $\Pi \cup \Pi_\approx$.

$$\langle x_1', x_2, x_3 \rangle \leftarrow \langle x_1, x_2, x_3 \rangle \wedge x_1 \approx x_1' \tag{$\approx_1$}$$
$$\langle x_1, x_2', x_3 \rangle \leftarrow \langle x_1, x_2, x_3 \rangle \wedge x_2 \approx x_2' \tag{$\approx_2$}$$
$$\langle x_1, x_2, x_3' \rangle \leftarrow \langle x_1, x_2, x_3 \rangle \wedge x_3 \approx x_3' \tag{$\approx_3$}$$
$$x_i \approx x_i \leftarrow \langle x_1, x_2, x_3 \rangle, \text{for } 1 \le i \le 3 \tag{$\approx_4$}$$

[1]http://www.cs.ox.ac.uk/isg/tools/RDFox/

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

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

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

3 Updating R-Materialisation Incrementally

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

$$\begin{aligned}
\Pi = \{\, & y_1 \approx y_2 \leftarrow \langle y_1, \mathsf{R}, x \rangle \wedge \langle y_2, \mathsf{R}, x \rangle, \\
& y_1 \approx y_2 \leftarrow \langle x, \mathsf{R}, y_1 \rangle \wedge \langle x, \mathsf{R}, y_2 \rangle \,\}
\end{aligned}$$

$$\begin{aligned}
E &= \{\, \langle \mathsf{a}, \mathsf{R}, \mathsf{b} \rangle, \langle \mathsf{c}, \mathsf{R}, \mathsf{d} \rangle, \langle \mathsf{a}, \mathsf{R}, \mathsf{d} \rangle \} \\
I &= \{\, \langle \mathsf{a}, \mathsf{R}, \mathsf{b} \rangle, \mathsf{a} \approx \mathsf{a}, \mathsf{R} \approx \mathsf{R}, \mathsf{b} \approx \mathsf{b}, \approx \approx \approx \} \\
\pi &= \{\, \mathsf{a} \mapsto \mathsf{a}, \mathsf{b} \mapsto \mathsf{b}, \mathsf{c} \mapsto \mathsf{a}, \mathsf{d} \mapsto \mathsf{b}, \mathsf{R} \mapsto \mathsf{R}, \approx \mapsto \approx \}
\end{aligned}$$

$$\begin{aligned}
E^- &= \{\, \langle \mathsf{a}, \mathsf{R}, \mathsf{d} \rangle \} \\
I' &= \{\, \langle \mathsf{a}, \mathsf{R}, \mathsf{b} \rangle, \mathsf{a} \approx \mathsf{a}, \mathsf{R} \approx \mathsf{R}, \mathsf{b} \approx \mathsf{b}, \approx \approx \approx, \\
&\qquad \langle \mathsf{c}, \mathsf{R}, \mathsf{d} \rangle, \mathsf{c} \approx \mathsf{c}, \mathsf{d} \approx \mathsf{d} \} \\
\pi' &= \{\, \mathsf{a} \mapsto \mathsf{a}, \mathsf{b} \mapsto \mathsf{b}, \mathsf{c} \mapsto \mathsf{c}, \mathsf{d} \mapsto \mathsf{d}, \mathsf{R} \mapsto \mathsf{R}, \approx \mapsto \approx \}
\end{aligned}$$

Relation R is bijective in Π, so $\mathsf{a} \approx \mathsf{c} \in J$ as both a and c have outgoing R-edges to d, and $\mathsf{b} \approx \mathsf{d} \in J$ as both b and d have incoming R-edges from a. By rewriting, we represent each fact $\langle \alpha, \mathsf{R}, \beta \rangle$ from J using a single fact $\langle \mathsf{a}, \mathsf{R}, \mathsf{b} \rangle$, and 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.

However, $\mathsf{a} \approx \mathsf{c} \notin J'$ and $\mathsf{b} \approx \mathsf{d} \notin J'$, so we must update π and extend I with the facts from J' that are not represented via π'. Thus, in our example, I' actually *contains* I.

Solution Overview. B/F$^{\approx}$ 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'$\triangleright$'.

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 I 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 π and I, which we apply to (π, I) in a final step. We next describe the components of B/F$^{\approx}$ in more detail.

Procedure $\mathsf{saturate}()$ 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^π; thus, C identifies the part of J' to recompute. Rather than storing L directly, we adapt the r-materialisation algorithm by Motik *et al.* (2015b) 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 $G \in F^\pi$; 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 σ of F to a body atom of r (line 48), we evaluate the remaining body atoms of r (line 49), and we derive $\gamma(\mathsf{h}(r)\tau)$ for each match τ (line 50). This basic idea is slightly more complicated by rewriting: if $F = a \approx b$, we modify γ so that one constant becomes the representative of the other one (line 45). As a consequence, facts can become 'outdated' w.r.t. γ, 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 ($\approx_4$) to F (line 47).

Procedure $\mathsf{saturate}()$ is repeatedly called in B/F$^{\approx}$. 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 $F \notin C^\pi$ and so we do not add $\gamma(F)$ to P, but C is later extended so that $F \in C^\pi$ holds. We handle this by maintaining a set Y of 'delayed' facts: in line 59 we add F to Y if $F \notin C^\pi$; and in line 40 we identify each 'delayed' fact $G \in C^\pi \cap Y$ and add $\gamma(G)$ to P.

Procedure $\mathsf{rewrite}(a, b)$ implements rewriting: we update γ (line 52), apply the replacement rules ($\approx_1$)–($\approx_3$) to already processed facts containing 'outdated' constants (line 54), ensure that Γ is normal w.r.t. γ (line 56), and reapply the normalised rules (lines 57–58). Motik *et al.* (2015b) discuss in detail the issues related to rule updating and reevaluation.

Procedure $\mathsf{checkProvability}()$ takes a fact $F \in I$ and ensures that, for each $G \in F^\pi$, 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 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^π become provable (lines 24, 28, 31, and 35). Lines 25–24 handle the reflexivity rules ($\approx_4$): to check provability of $c \approx c$, we recursively check the provability each fact containing c. Lines 29–31 handle replacement rules ($\approx_1$)–($\approx_3$): we recursively check the provability of $c \approx c$ for each constant c occurring in F. Finally, lines 32–35 handle the rules in $\pi(\Pi)$: we consider each rule $r \in \pi(\Pi)$ whose head matches F and each substitution τ that matches the body of r in I, and we recursively check the provability of $\mathsf{b}(r)\tau$.

Procedure $\mathsf{BF}^{\approx}()$ 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 $G \in F^\pi$ 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 $\pi(\Pi)$: we identify each rule $r \in \pi(\Pi)$ where F matches a body atom of r, we evaluate the remaining body atoms of r in I, and we add $\mathsf{h}(r)\tau$ to D for each τ such that $\mathsf{b}(r)\tau \subseteq I$. Once D is processed, $(\gamma, P \setminus \hat{P})$ reflects the changes to (π, 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 ($\approx_4$), in lines 25–28 we may end up recursively proving each fact G that mentions c. However, F holds trivially if E' 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 Γ contains a rule $y_1 \approx y_2 \leftarrow \langle x, \mathsf{R}, y_1 \rangle \wedge \langle x, \mathsf{R}, y_2 \rangle$, and consider a call to $\mathsf{saturate}()$ 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 $\langle x, \mathsf{R}, y_1 \rangle$, and once when we match $\langle a, R, d \rangle$ to $\langle x, \mathsf{R}, y_2 \rangle$. 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 $\langle a, R, d \rangle \notin V$, so τ_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, τ_2 is thus considered only when $\langle a, R, b\rangle$ is matched to $\langle x, \mathsf{R}, y_1\rangle$.

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 $F^\pi \cap J' = \emptyset$, no fact in F^π 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 S from backward chaining.

Singletons. If we encounter $F = c \approx c$ in line 9 or 29 where c represents only itself (i.e., $|c^\pi| = 1$), then we know that no fact in F^π 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.* (2015c) to formalise B/F$^{\approx}$. We recapitulate some definitions, present the pseudocode, and formally state the algorithm's properties.

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

An *annotated query* has the form $Q = B_1^{\bowtie_1} \wedge \cdots \wedge B_k^{\bowtie_k}$, where each B_i is an atom and *annotation* $\bowtie_i$ is either empty or equal to $\neq$. Given datasets X and Y and a substitution σ, operation $X.\mathsf{eval}(Q, Y, \sigma)$ returns a set containing each smallest substitution τ such that $\sigma \subseteq \tau$ and, for $1 \leq i \leq k$, (i) $B_i\tau \in X$ if $\bowtie_i$ is empty or (ii) $B_i\tau \in X \setminus Y$ if $\bowtie_i$ is $\neq$. 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 $\Pi.\mathsf{matchHead}(F)$ returns all tuples $\langle r, Q, \sigma\rangle$ with $r \in \Pi$ a rule of the form (1), σ a substitution such that $H\sigma = F$, and $Q = B_1 \wedge \cdots \wedge B_n$. Moreover, operation $\Pi.\mathsf{matchBody}(F)$ returns all tuples $\langle r, Q, \sigma\rangle$ with $r \in \Pi$ a rule of the form (1), σ a substitution such that $B_i\sigma = F$ for some $1 \leq i \leq n$, and Q is defined as

$$Q = B_1^{\neq} \wedge \cdots \wedge B_{i-1}^{\neq} \wedge B_{i+1} \wedge \cdots \wedge B_n. \qquad (2)$$

Finally, given a mapping γ of constants to constants, and constants d and c, operation $\gamma.\mathsf{mergeInto}(d, c)$ modifies γ 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 seminaïve algorithm [Abiteboul *et al.*, 1995], it does not repeat derivations; the proof is given in a technical report [Motik *et al.*, 2015a].

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 E. *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 ($|I^{\approx}|$), and the time ($T^{\approx}$) and the number of derivations ($D^{\approx}$) used to compute it via 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 $\Delta|I^{\approx}|$ and $\Delta|I^A|$ 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 $|c^\pi| = 3930$; thus, when $\approx$ is axiomatised, deriving just all equalities involving c^π requires $3930^3 = 60$ billion derivations, which causes the initial materialisation to last longer than four hours. The number of derivations D in

[2]https://krr-nas.cs.ox.ac.uk/2015/IJCAI/RDFox/index.html

[3]http://www.clarosnet.org/XDB/ASP/clarosHome/

[4]http://dbpedia.org/

[5]http://www.uniprot.org

[6]http://www.cyc.com/platform/opencyc

Input Variables

E	: the explicit facts
Π	: the datalog program
(π, I)	: the r-materialisation of E w.r.t. Π
E^-	: the facts to delete from E

Global Temporary Variables

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

Algorithm 1 B/F$^{\approx}$()

```
*  1: $C := D := P := \hat{P} := Y := O := S := V := \emptyset$
▷  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 $F = c \approx c$ and $|c^\pi| > 1$ then
▷ 10:             for each $G \in I \setminus O$ with $c \in$ voc($G$) do
▷ 11:                 $D$.add($G$)
▷ 12:         for each $c \in$ voc($F$) do $D$.add($c \approx c$)
  13:         for each $\langle r, Q, \sigma \rangle \in \pi(\Pi)$.matchBody($F$) do
  14:             for each $\tau \in [I \setminus O]$.eval($Q, \{F\}, \sigma$) do
  15:                 $D$.add(h($r$)$\tau$)
  16:         $O$.add($F$)
* 17: propagateChanges()
```

▷ **Algorithm 2** propagateChanges()

```
18: for each $c \approx c \in C$ 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($F$):
    t iff $F \notin S$ and $\gamma(F^\pi) \subseteq (P \setminus \hat{P})$
allDisproved($F$):
    t iff $\gamma(F^\pi) \cap (P \setminus \hat{P}) = \emptyset$
```

Algorithm 4 checkProvability(F)

```
  22: if not $C$.add($F$) then return
  23: saturate()
* 24: if allProved($F$) then return
▷ 25: if $F = c \approx c$ then
▷ 26:     for each $G \in I \setminus S$ with $c \in$ voc($G$) do
▷ 27:         checkProvability($G$)
▷ 28:         if allProved($F$) then return
▷ 29: for each $c \in$ voc($F$) with $c \approx c \notin S$ and $|c^\pi| > 1$ do
▷ 30:     checkProvability($c \approx c$)
▷ 31:     if allProved($F$) then return
  32: for each $\langle r, Q, \sigma \rangle \in \pi(\Pi)$.matchHead($F$) do
  33:     for each $\tau \in [I \setminus S]$.eval($Q, \emptyset, \sigma$) and $G \in$ b($r$)$\tau$ do
  34:         checkProvability($G$)
  35:         if allProved($F$) then return
```

Algorithm 5 saturate()

```
  36: while ($F := C$.next) $\neq \varepsilon$ do
▷ 37:     if $F = c \approx c$ then
▷ 38:         for each $d \in$ voc($E$) with $\pi(d) = c$ do
▷ 39:             $P$.add($\gamma(d) \approx \gamma(d)$)
* 40:     for each $G \in F^\pi \cap (E \cup Y)$ do $P$.add($\gamma(G)$)
  41: while ($F := P$.next) $\neq \varepsilon$ do
* 42:     if $F \in P \setminus (\hat{P} \cup V)$ and $V$.add($F$) then
▷ 43:         $G := \gamma(F)$
▷ 44:         if $F \neq G$ then $\hat{P}$.add($F$) and $P$.add($G$)
▷ 45:         else if $F = a \approx b$ and $a \neq b$ then rewrite($a, b$)
▷ 46:         else
* 47:             for each $c \in$ voc($G$) do prove($c \approx c$)
  48:             for each $\langle r, Q, \sigma \rangle \in \Gamma$.matchBody($G$) do
* 49:                 for each $\tau \in [V \setminus \hat{P}]$.eval($Q, \{G\}, \sigma$) do
* 50:                     prove(h($r$)$\tau$)
```

▷ **Algorithm 6** rewrite(a, b)

```
51: $c := \min\{a, b\}$     $d := \max\{a, b\}$
52: $\gamma$.mergeInto($d, c$)
53: for each $F \in P \setminus \hat{P}$ with $d \in$ voc($F$) do
54:     $\hat{P}$.add($F$) and $P$.add($\gamma(F)$)
55: for each $r \in \Gamma$ with $r \neq \gamma(r)$ do
56:     replace $r$ in $\Gamma$ with $r' := \gamma(r)$
57:     for each $\tau \in [V \setminus \hat{P}]$.eval(b($r'$), $\emptyset, \emptyset$) do
58:         prove(h($r'$)$\tau$)
```

▷ **Algorithm 7** prove(F)

```
59: if $\pi(F) \in C$ then $P$.add($F$) else $Y$.add($F$)
```

Theorem 1. *Let (π, I) be the r-materialisation of a dataset E w.r.t. a program Π, and let E^- be a dataset.*

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

UOBM-100-L $|E| = 24.5\text{M}$ $|I^{\approx}| = 46.4\text{M}$ $T^{\approx} = 69$ $D^{\approx} = 79.3\text{M}$
$|\Pi| = 210$ $|I^A| = 46.7\text{M}$ $T^A = 122$ $D^A = 361\text{M}$

$\lvert E^-\rvert$	$\Delta\lvert I^{\approx}\rvert$	B/F$^{\approx}$		Remat$^{\approx}$		$\Delta\lvert I^A\rvert$	B/F^A		Remat^A	
		T	D	T	D		T	D	T	D
100	146	0.6	0.7k	45.1	79.3M	146	4.6	32.8k	94.6	361M
5k	7.8k	1.2	45.8k	42.5	79.3M	7.9k	7.1	805k	93.1	361M
1.3M	1.9M	18.2	8.7M	39.2	75.4M	2.0M	38.0	98.0M	89.5	361M
2.5M	3.9M	29.9	15.8M	41.7	71.5M	4.0M	54.5	151M	83.7	345M
3.8M	5.8M	31.8	22.3M	37.4	67.7M	5.9M	70.9	188M	79.3	329M
5M	7.7M	41.2	28.4M	36.2	63.8M	7.9M	73.8	218M	73.2	314M

UOBM-100-U $|E| = 24.5\text{M}$ $|I^{\approx}| = 78.8\text{M}$ $T^{\approx} = 225$ $D^{\approx} = 719\text{M}$
$|\Pi| = 279$ $|I^A| = —$ $T^A = —$ $D^A = —$

$\lvert E^-\rvert$	$\Delta\lvert I^{\approx}\rvert$	B/F$^{\approx}$		Remat$^{\approx}$		$\Delta\lvert I^A\rvert$	B/F^A		Remat^A	
		T	D	T	D		T	D	T	D
100	197	3.5	21.5k	209	719M	—	—	—	—	—
1k	1.8k	277	581M	219	719M	—	—	—	—	—
2.5k	4.3k	338	584M	209	719M	—	—	—	—	—
5k	8.5k	345	584M	214	719M	—	—	—	—	—

Claros-L $|E| = 18.8\text{M}$ $|I^{\approx}| = 79.5\text{M}$ $T^{\approx} = 83$ $D^{\approx} = 129\text{M}$
$|\Pi| = 1.3\text{k}$ $|I^A| = 102\text{M}$ $T^A = 3.9\text{k}$ $D^A = 11.0\text{G}$

$\lvert E^-\rvert$	$\Delta\lvert I^{\approx}\rvert$	B/F$^{\approx}$		Remat$^{\approx}$		$\Delta\lvert I^A\rvert$	B/F^A		Remat^A	
		T	D	T	D		T	D	T	D
100	209	8.3	797k	77.4	135M	819	2476	15.8G	3174	11.0G
5k	11.2k	9.1	895k	77.0	135M	18.6k	2609	15.8G	3166	11.0G
750k	1.7M	29.5	14.5M	80.9	131M	4.0M	2816	17.1G	2690	9.5G
1.5M	3.5M	46.1	26.5M	81.5	127M	10.1M	2757	17.4G	1933	7.3G
2.3M	5.3M	63.9	38.4M	77.7	123M	15.3M	3092	18.3G	1389	5.5G
3M	7.2M	78.4	48.8M	72.4	119M	19.4M	3170	18.6G	1075	4.4G

Claros-LE $|E| = 18.8\text{M}$ $|I^{\approx}| = 539\text{M}$ $T^{\approx} = 4514$ $D^{\approx} = 12.6\text{G}$
$|\Pi| = 1.3\text{k}$ $|I^A| = 562\text{M}$ $T^A = 9048$ $D^A = 26.3\text{G}$

$\lvert E^-\rvert$	$\Delta\lvert I^{\approx}\rvert$	B/F$^{\approx}$		Remat$^{\approx}$		$\Delta\lvert I^A\rvert$	B/F^A		Remat^A	
		T	D	T	D		T	D	T	D
100	522	16.1	617k	4397	12.6G	1132	5703	25.8G	8693	26.3G
2.5k	179k	31.6	9.9M	4430	12.6G	—	—	—	—	—
5k	427k	39.4	10.7M	4392	12.6G	435k	5845	25.8G	9383	26.3G
7.5k	609k	44.8	11.6M	4713	12.6G	—	—	—	—	—
10k	781k	4300	12.4G	4627	12.6G	—	—	—	—	—

DBpedia-L $|E| = 113\text{M}$ $|I^{\approx}| = 136\text{M}$ $T^{\approx} = 49.3$ $D^{\approx} = 36.6\text{M}$
$|\Pi| = 3.4\text{k}$ $|I^A| = 139\text{M}$ $T^A = 641$ $D^A = 895\text{M}$

$\lvert E^-\rvert$	$\Delta\lvert I^{\approx}\rvert$	B/F$^{\approx}$		Remat$^{\approx}$		$\Delta\lvert I^A\rvert$	B/F^A		Remat^A	
		T	D	T	D		T	D	T	D
100	105	0.3	91	47.5	36.6M	105	8.9	1.7M	251	895M
5k	5.0k	0.4	24.4k	64.6	36.6M	5.3k	20.3	5.7M	256	895M
1.8M	1.8M	29.4	2.1M	48.7	36.3M	2.0M	50.0	72.2M	239	895M
3.5M	3.6M	38.9	3.6M	49.0	35.9M	3.9M	85.5	116M	237	881M
5.3M	5.3M	52.2	4.9M	54.3	35.5M	5.9M	89.8	152M	232	866M
7M	7.1M	63.1	6.2M	50.7	35.1M	7.8M	103	184M	227	852M

UniProt-L $|E| = 123\text{M}$ $|I^{\approx}| = 179\text{M}$ $T^{\approx} = 118$ $D^{\approx} = 183\text{M}$
$|\Pi| = 451$ $|I^A| = 229\text{M}$ $T^A = 527$ $D^A = 1.6\text{G}$

$\lvert E^-\rvert$	$\Delta\lvert I^{\approx}\rvert$	B/F$^{\approx}$		Remat$^{\approx}$		$\Delta\lvert I^A\rvert$	B/F^A		Remat^A	
		T	D	T	D		T	D	T	D
100	125	2.5	892	235	238M	125	14.3	6.0k	490	1.6G
5k	6.1k	3.4	35k	221	238M	6.1k	17.5	271k	482	1.6G
4.5M	5.7M	84.0	24.8M	204	232M	5.7M	125	190M	475	1.5G
9M	11.5M	137	46.7M	216	225M	11.5M	192	344M	478	1.5M
13.5M	17.4M	209	67.1M	220	218M	17.4M	315	483M	473	1.4G
18M	23.4M	220	86.5M	217	210M	23.4M	371	613M	481	1.4G

OpenCyc-L $|E| = 2.4\text{M}$ $|I^{\approx}| = 141\text{M}$ $T^{\approx} = 164$ $D^{\approx} = 280\text{M}$
$|\Pi| = 261\text{k}$ $|I^A| = 1.2\text{G}$ $T^A = 3.5\text{k}$ $D^A = 12.9\text{G}$

$\lvert E^-\rvert$	$\Delta\lvert I^{\approx}\rvert$	B/F$^{\approx}$		Remat$^{\approx}$		$\Delta\lvert I^A\rvert$	B/F^A		Remat^A	
		T	D	T	D		T	D	T	D
100	5.4k	15.5	405k	220	280M	50.0k	472	8.5M	3296	12.9G
1k	53.1k	1062	69.5M	222	280M	5.1M	5537	2.0G	3479	12.9G
2.5k	130k	1078	69.8M	178	279M	5.8M	5339	2.1G	3621	12.8G
5k	261k	1123	70.4M	177	279M	7.2M	5475	2.1G	3334	12.8G

Table 1: Experimental results

B/F$^{\approx}$ 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$^{\approx}$ outperforms all other approaches, often by orders of magnitude, and in most cases it does so even for much larger updates.

Even when $|I^A| - |I^{\approx}|$ is 'small' (i.e., when not many equalities are derived), B/F$^{\approx}$ outperforms B/F^A. This seems to be mainly because B/F^A ascribes no special meaning to $\Pi_{\approx}$ and so it does not use the optimisation from lines 37–39; thus, when trying to prove $c \approx c$, B/F^A performs backward chaining via rules ($\approx_4$) and so it potentially examines each fact containing c. On Claros-L, although $|I^A|$ and $|I^{\approx}|$ are of similar sizes, I^A contains one constant c with $|c^{\pi}| = 306$, which gives rise to 306^3 derivations; this explains the difference in the performance of B/F$^{\approx}$ and B/F^A.

Remat$^{\approx}$ outperforms B/F$^{\approx}$ in cases similar to those described by Motik *et al.* (2015c). For example, in UOBM, relation hasSameHomeTownWith is symmetric and transitive, which creates cliques of connected constants; B/F always recomputes each changed clique, thus repeating most of the 'hard' work. Equality connects constants in cliques, which poses similar problems for B/F$^{\approx}$. For example, due to the constant c with $|c^{\pi}| = 3930$, deleting 5k facts in UOBM-100-U results in only 961k (about 1.2% of $|I^{\approx}|$) 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$^{\approx}$ repeats in Algorithm 5 a substantial portion of the initial work.

On OpenCyc-L, Remat$^{\approx}$ already outperforms B/F$^{\approx}$ 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 $\langle x, \text{type}, y\rangle \leftarrow \langle x, R_i, y\rangle$ that never fire during forward chaining; however, to check provability of $\langle a, \text{type}, C\rangle$, Algorithm 4 considers in line 32 each time each of the 200 rules. After removing all such 'idle' rules manually, B/F$^{\approx}$ and Remat$^{\approx}$ could update 1k tuples in roughly the same time. Further analysis revealed that the slowdown in B/F$^{\approx}$ occurs mainly in line 40: the condition is checked for 13.3M facts F, and these give rise to 139M facts in F^{π}, 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.

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^{π}. 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^3, Score!, and DBOnto, and the FP7 project Optique.

References

[Abiteboul *et al.*, 1995] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[Aref, 2010] Molham Aref. Datalog for Enterprise Software: from Industrial Applications to Research (Invited Talk). In *Tech. Comm. ICLP*, volume 7, page 1, 2010.

[Baader and Nipkow, 1998] F. Baader and T. Nipkow. *Term Rewriting and All That*. CUP, 1998.

[Bishop *et al.*, 2011] Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1):33–42, 2011.

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

[Dewan *et al.*, 1992] H. M. Dewan, D. Ohsie, S. J. Stolfo, O. Wolfson, and S. Da Silva. Incremental Database Rule Processing In PARADISER. *Journal of Intelligent Information Systems*, 1(2):177–209, 1992.

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

[Goasdoué *et al.*, 2013] François Goasdoué, Ioana Manolescu, and Alexandra Roatis. Efficient query answering against dynamic RDF databases. In *Proc. EDBT*, pages 299–310. ACM, 2013.

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

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

[Gupta *et al.*, 1993] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining Views Incrementally. In *Proc. SIGMOD*, pages 157–166. ACM, 1993.

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

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

[Motik *et al.*, 2009] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language: Profiles, W3C Recommendation, October 27 2009.

[Motik *et al.*, 2014] Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In *Proc. AAAI*, 2014.

[Motik *et al.*, 2015a] Boris Motik, Yavor Nenov, Robert Piro, and Ian Horrocks. Combining rewriting and incremental materialisation maintenance for datalog programs with equality. *CoRR*, 2015.

[Motik *et al.*, 2015b] Boris Motik, Yavor Nenov, Robert Piro, and Ian Horrocks. Handling owl:sameAs via Rewriting. In *Proc. AAAI*, 2015.

[Motik *et al.*, 2015c] Boris Motik, Yavor Nenov, Robert Piro, and Ian Horrocks. Incremental Update of Datalog Materialisation: the Backward/Forward Algorithm. In *Proc. AAAI*, 2015.

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

[Nieuwenhuis and Rubio, 2001] R. Nieuwenhuis and A. Rubio. Paramodulation-Based Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.

[Stocker and Smith, 2008] Markus Stocker and Michael Smith. Owlgres: A Scalable OWL Reasoner. In *Proc. OWLED*, 2008.

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

[Urbani *et al.*, 2013] J. Urbani, A. Margara, C. J. H. Jacobs, F. van Harmelen, and H. E. Bal. DynamiTE: Parallel Materialization of Dynamic RDF Data. In *Proc. ISWC*, volume 8218, pages 657–672. Springer, 2013.

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

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