Embracing Change by Abstraction
Materialization Maintenance for Large ABoxes

Markus Brenner and Birte Glimm
University of Ulm, Germany
markus.brenner@uni-ulm.de, birte.glimm@uni-ulm.de

Abstract
Abstraction Refinement is a technique which allows for reducing materialization of an ontology with a large ABox to materialization of a smaller (compressed) ‘abstraction’ of this ontology. In this paper, we show how Abstraction Refinement can be adopted for incremental ABox materialization by combining it with the well-known DRed algorithm for materialization maintenance. The combination is non-trivial and to preserve correctness, already Horn ALC\H\I requires more complex abstractions. Nevertheless, we show that significant benefits can be obtained for synthetic and real-world ontologies.

1 Introduction
Most ontology reasoners support the task of materialization (i.e., they compute and explicitly store all entailed atomic concept and role assertions for the individuals in the ontology), which allows for directly evaluating conjunctive instance queries. Computing the materialization is computationally expensive and approaches such as summarization [Dolby et al., 2009], ABox modularization [Wandelt and Möller, 2012], or Abstraction Refinement [Glimm et al., 2014; 2017] attempt to “compress” the size of the dataset over which the materialization is computed to be able to materialize large ABoxes. Even using efficient materialization techniques, recomputing all consequences whenever input data changes can cause a significant delay before user queries can be answered again, which might be prohibitive for some application scenarios. Incremental maintenance algorithms originating from the database and Datalog communities (see, e.g., [Volz et al., 2003; Motik et al., 2013]) have been applied to description logics and the semantic web for incremental classification [Kazakov and Klinov, 2013], incremental materialization via Datalog [Volz et al., 2005] and RDF stream reasoning [Barbieri et al., 2010]. Apart from the technique used in the RDFox system [Motik et al., 2014], the mentioned materialization algorithms focus on and are optimised for ontologies that can be expressed in the form of (Datalog) rules, i.e., proper existentials are not supported. Furthermore, how incremental maintenance algorithms can be combined with data compression techniques is an open problem, only addressed in a sketchy way by Steigmiller et al. [Steigmiller et al., 2015] in the form of a representative cache maintaining individuals in an incremental fashion.

We address this problem by extending the Abstraction Refinement method s.t. abstractions capture information beyond the simple presence of assertions. This ultimately allows us to deal with changes of the underlying ABox. To show the correctness of the obtained theoretical framework, we devise novel proof techniques. Our empirical evaluation with synthetic and real-world ontologies shows up to four times improved materialization times compared to the approach without Abstraction Refinement in different change scenarios.

2 Preliminaries
The syntax of ALC\H\I is defined using a vocabulary consisting of countably infinite disjoint sets NC of atomic concepts, NR of atomic roles, and NI of individuals. A role is either an atomic role or an inverse role r− with r ∈ NR. We denote atomic roles with lower case and (possibly non-atomic) roles with upper case letters. We define R− := r− if r = r and R− := r−. An ALC\H\I concept is defined as
\[ C ::= \top | \bot | A | \neg C | C_1 \sqcap C_2 | C_1 \sqcup C_2 | \forall R.C | \exists R.C, \]
where A ∈ NC and R is a role. Let C, D be concepts, R, S roles and a, b individuals. A TBox is a set of concept and role inclusion axioms of the form C ⊑ D and R ⊑ S, respectively. An ABox is a set of (concept and role) assertions of the form C(a) and R(a, b), respectively. An ontology O = A∪T consists of an ABox and a TBox. To simplify the presentation, we do not distinguish between R(a, b) and R−(b, a) as well as R ⊑ S and R− ⊑ S−. We use con(O), rol(O), and ind(O) for the sets of atomic concepts, atomic roles, and individuals occurring in O, respectively.

An interpretation I = (ΔI, τI) consists of a non-empty set ΔI, the domain of I, and an interpretation function τI, that assigns to each A ∈ NC a subset ΔI ⊆ ΔI, ⊇ ΔI to each R ∈ NR a binary relation RΔI ⊆ ΔI × ΔI, and to each a ∈ NI an element aΔI ∈ ΔI. This assignment is extended to roles by (r−)ΔI = {⟨ d, e ⟩ | ⟨ d, e ⟩ ∈ rΔI } and, inductively, to complex concepts as τI = ΔD, ΔI = ΔD, (¬C)ΔI = ΔI \ ΔD, (C ∩ D)ΔI = ΔI ∩ ΔD, (C ∪ D)ΔI = ΔI ∪ ΔD, and
\[ (\forall r.C)ΔI = \{ d ∈ ΔI | ⟨ d, e ⟩ ∈ RΔI \rightarrow e ∈ CΔI \}, \]
\[ (\exists r.C)ΔI = \{ d ∈ ΔI | e ∈ CΔI : ⟨ d, e ⟩ ∈ RΔI \}. \]
An interpretation I satisfies an axiom α, written I |= α, if
\[ \alpha = C \subseteq D \text{ and } C^T \subseteq D^T, \alpha = R \subseteq S \text{ and } R^T \subseteq S^T, \]

\[ \alpha = C(a) \text{ and } a^T \subseteq C^T, \text{ or } \alpha = R(a, b) \text{ and } \langle a^T, b^T \rangle \in R^T; \]

\[ \mathcal{I} \text{ is a model of } \alpha \text{, written } \mathcal{I} \models \alpha, \text{ if } \mathcal{I} \models \alpha \text{ for each } \alpha \in \mathcal{O}; \mathcal{O} \text{ entails an axiom } \alpha, \text{ written } \mathcal{O} \models \alpha, \text{ if every model of } \mathcal{O} \text{ satisfies } \alpha. \]

An \textit{ALCHI} ontology \( \mathcal{O} \) is \textit{Horn} [Krötzsch et al., 2013] and in normalized form if, for every \( \alpha \in \mathcal{O}, \mathcal{C} \in \mathcal{N}_C \) and, every \( C \subseteq D \in \mathcal{O} \), is in one of the forms \( T \sqsubseteq A, A \sqsubseteq B, A \sqsubseteq \exists R.B, A \sqsubseteq \exists R.C, A \sqsubseteq \forall R.B \) where \( A, B, C \in \mathcal{N}_C \) and \( R \) is a role. W.l.o.g., we assume in the remainder every ontology as normalized by applying a structural transformation; see e.g., [Kazakov, 2009].

For \textit{ALCHI} ontologies, determining entailed role assertions can easily be done using a precomputed role hierarchy. Hence, we focus on the task of computing entailed concept assertions: For an ontology \( \mathcal{O} = A \cup T \), we say that \( A \) is \textit{materialized} w.r.t. \( T \), if \( \mathcal{O} \models A \) implies \( A(a) \in A \) for each \( A \in \text{con}(\mathcal{O}) \) and \( a \in \text{ind}(\mathcal{O}) \). The materialization of \( A \) w.r.t. \( T \) is the smallest super-set of \( A \) that is materialized w.r.t. \( T \).

### 2.1 Abstraction Refinement

The main idea of the Abstraction Refinement method [Glimm et al., 2014; 2017] is to materialize an ontology \( \mathcal{O} = A \cup T \) with a potentially large ABox \( A \) by constructing a smaller ABox \( B \) s.t. the materialization of \( B \cup T \) can be computed by a general-purpose reasoner, and transferring the new entailments back to \( \mathcal{O} \). The ABox \( B \) is usually called the abstraction of the original ABox \( A \). Homomorphisms are used for transferring back entailments:

**Definition 1.** For an \( A \) and \( B \) ABoxes, a mapping \( h: \text{ind}(B) \to \text{ind}(A) \) is called a homomorphism (from \( B \) to \( A \)) if \( h(B) = \cup_{a \in h(a)} \subseteq A \), where \( h(C(a)) = C(h(a)) \) and \( h(R(a, b)) = h(R(h(a), h(b))) \). An individual \( b \in \text{ind}(B) \) is a representative of an individual \( a \in \text{ind}(A) \) if there exists a homomorphism \( h: \text{ind}(B) \to \text{ind}(A) \) s.t. \( h(b) = a \).

The following property enables the transfer of entailments:

**Lemma 1.** Let \( h: \text{ind}(B) \to \text{ind}(A) \) be a homomorphism between the ABoxes \( B \) and \( A \). Then, for every \( \mathcal{T} \) and every axiom \( \beta, B \cup T \models \beta \) implies \( A \cup T \models h(\beta) \).

Abstractions are based on asserted roles and concepts for single individuals using types:

**Definition 2 (Type).** Let \( A \) be an ABox and \( a \) an individual. The concept type of \( a \) w.r.t. \( A \) is a set of concepts \( \gamma_C(a) = \{ C \mid C(a) \in A \} \). The role type of \( a \) w.r.t. \( A \) is a set of roles \( \gamma_R(a) = \{ R \mid \exists b : R(b, a) \in A \} \). The (combined) type of \( a \) w.r.t. \( A \) is a pair \( (\gamma_C(a), \gamma_R(a)) \), where \( \gamma_C(a) \) and \( \gamma_R(a) \) are the concept and role type of \( a \) w.r.t. \( A \), respectively.

**Example 1.** Let \( A = \{ A(a), A(b), R(a, b) \} \). Then \( \gamma_C(a) = \{ A \}, \gamma_R(a) = \{ \} \), \( \gamma_C(b) = \{ A \}, \gamma_R(b) = \{ \} \), \( \gamma_C(a) = \{ A \}, \gamma_R(a) = \{ \} \), and \( \gamma_C(b) = \{ A \}, \gamma_R(b) = \{ \} \).

The abstract ABox is then constructed by introducing one representative for each type with the respective assertions.

**Definition 3 (Abstraction).** Let \( \tau = (\gamma_C, \gamma_R) \) be a type. The abstraction for \( \tau \) is an ABox

\[ B_\tau = \{ C(v_\tau) \mid C \in \gamma_C \} \cup \{ R(v_\tau, w^R_\tau) \mid R \in \gamma_R \}, \]

where \( v_\tau \) and \( w^R_\tau \) are distinguished abstract individuals for the type \( \tau \). The abstraction of an ABox \( A \) is \( \bigcup_{a \in \text{ind}(A)} B_{\tau(a)} \), where \( \tau(a) \) is the type for a w.r.t. \( A \).

**Example 2.** The abstraction for \( A \) in Example 1 is \( B = B_{\tau(a)} \cup B_{\tau(b)} \), where \( B_{\tau(a)} = \{\{ A(v_{\tau_1}), r(v_{\tau_1}, w_{\tau_1}^r) \}\}, B_{\tau(b)} = \{\{ A(v_{\tau_2}), r(v_{\tau_2}, w_{\tau_2}^r) \}\} \).

Intuitively, the abstraction is a disjoint union of ABoxes simulating combined types. Note that each mapping \( h: \text{ind}(B) \to \text{ind}(A) \) s.t. \( h(v_\tau) \in \{ a \in \text{ind}(A) \mid \tau(a) = \tau \} \), and \( h(w_\tau^r) \in \{ b \in \text{ind}(A) \mid \exists \tau(b) \} \) is a homomorphism from \( B \) to \( A \), which allows for transferring entailments from the abstraction back to the original ABox (Lemma 1). For \( B^\infty \) the materialization of \( B \cup T \) and \( \Delta B = B^\infty \setminus B \), the function \( \text{update} (\Delta B, A) \) uses the above defined homomorphism to generate an update using \( \Delta B \).

Algorithm 1 then defines the Abstraction Refinement method for materializing a Horn \textit{ALCHI} ontology. The loop is repeated until a fix-point is reached and the update contains no assertions not already present in the materialization. The method is sound, complete, and terminating for Horn \textit{ALCHI} [Glimm et al., 2014] and, with some extensions, even for Horn \textit{SHOIF} [Glimm et al., 2017].

### 3 Incremental Materialization

For computing the materialization, we compute the closure of the ABox assertions using a modified version of the materialization rules given by Glimm et al. (2017) for Horn \textit{SHOIF} restricted to Horn \textit{ALCHI} as shown in Figure 1. Premises are given above the horizontal line and the conclusions below. Side conditions are given after the colon and restrict the expressions to which the rules are applicable. For example, rule
\[ R_\mathcal{C} : \frac{A_1(a) \cdots A_n(a)}{B(a)} : \mathcal{T} \models \bigwedge_i A_i \subseteq B \]
\[ R_\mathcal{V} : \frac{A_1(a) \cdots A_n(a) R(a,b)}{B(b)} : \mathcal{T} \models \bigwedge_i A_i \equiv \forall S.B \quad \text{and} \quad \mathcal{T} \models R \subseteq S \]

Figure 1: Materialization rules \((A, B \in N_c, R, S \text{ roles}, a, b \in N_i)\)

\( R_{\mathcal{C}} \) produces one inference for each individual \( a \) and concepts \( A_1, \ldots, A_n, B \) s.t. \( \mathcal{T} \models A_0 \supseteq \cdots \supseteq A_n \subseteq B \) with the premise \( \{A_1(a), \ldots, A_n(a)\} \) and the conclusion \( B(a) \). Note that TBox axioms are only used in side conditions and never as premises of the rules, which allows us to focus on ABox reasoning and leave TBox reasoning to a suitable reasoner. Theorem 1 can be shown in a very similar (but simpler) way to the proof used by Glimm et al. (2017).

**Theorem 1.** The rules in Figure 1 are sound and complete for the materialization of a normalized Horn \( \text{ALCHI} \) ontology.

### 3.1 The Delete/Rederive Algorithm

For the changes, we adopt the database view, in which only known ABox facts can be added or deleted. This already provides enough expressivity for many use cases, e.g. stream reasoning [Barbieri et al., 2010]. Due to the monotonicity of DL reasoning, additions can be handled by simply ‘continuing’ materialization. Hence, we only focus on deletions and assume: Given a Horn \( \text{ALCHI} \) ontology \( \mathcal{O} = \mathcal{A} \cup \mathcal{T} \), the materialization \( \mathcal{A}^\infty = \mathcal{A} \cup \mathcal{T} \) w.r.t. \( \mathcal{T} \), and a set \( \mathcal{A}^- \subseteq \mathcal{A} \) of deletions, we want to determine the materialization of \( \mathcal{A} \setminus \mathcal{A}^- \) w.r.t. \( \mathcal{T} \) using \( \mathcal{A}^\infty \). This is non-trivial and requires the identification and removal of assertions in \( \mathcal{A}^\infty \) no longer derivable from \( \mathcal{A} \setminus \mathcal{A}^- \). The Delete/Rederive (DRed) algorithm [Gupta et al., 1993; Staudt and Jarke, 1996] initially overestimates the necessary deletions and then determines facts still derivable from \( \mathcal{A} \setminus \mathcal{A}^- \). The overestimation is obtained by continuously (over-)deleting facts, which can be derived using already deleted facts. We formalize this using restricted rule applications.

**Definition 4 (Restricted Derivation).** Given a set of deduction rules \( \mathcal{R} \), an ontology \( \mathcal{O} = \mathcal{A} \cup \mathcal{T} \), and an ABox \( \mathcal{A}' \), an axiom \( \alpha \) can directly be derived from \( \mathcal{O} \) under restriction \( \mathcal{A}' \), written \( \mathcal{O} \vdash_{\mathcal{A}'} \alpha \), if \( \alpha \) can be derived from \( \mathcal{O} \) by a rule in \( \mathcal{R} \) s.t. at least one of the premises is in \( \mathcal{A}' \).

In the remainder, we simply write \( \mathcal{A} \cup \mathcal{T} \vdash_{\mathcal{A} \setminus \mathcal{A}^-} \alpha \) and assume \( \mathcal{R} \) to consist of the rules from Figure 1.

For the presentation of the DRed algorithm, we follow the presentation style of Motik et al. (2015), but we adapt it to a consequence-based calculus, to avoid complications with ensuring termination of a Datalog program in the presence of function symbols, which are required to handle existential quantifiers. DRed, formalized in Algorithm 2, consists of three consecutive phases: (1) compute the overdeletion, (2) rederive deletions with an alternative derivation in one step, (3) reinsert facts with alternative derivations. Phase (1) and (3) can directly be used to extend the Abstraction Refinement method for incremental materialization maintenance. Hence, we define the according functions overdelete and reinsert, which are then used in Algorithm 2. The computed set of deletions \( D_{\text{all}} \) in the function overdelete can be seen as a sort of closure for \( \mathcal{A}^\infty \cup \mathcal{T} \). The search for facts with an alternative derivation in the reinsertion phase is restricted to assertions in the overdeletion set \( D_{\text{all}} \) using an additional premise for the derivation rules. The computed set \( D_{\text{new}} \) is then used in Phase (3) by the function reinsert to restore the correct materialization. Rule applicability is restricted to avoid unnecessary rule applications. Although \( D_{\text{all}} \) does not appear in the reinsertion phase, due to the way the overdeletion is generated, Phase (3) determines \( \mathcal{A}_{\text{new}}^\infty \) as a closure for \( \mathcal{A}_{\text{new}}^\infty \cup \mathcal{T} \).

**3.2 DRed and Abstraction Refinement**

We adopt DRed in the general Abstraction Refinement way: We construct, for each of the different phases, suitable abstractions of the ABox on which we perform the respective operations. Interleaved refinement steps (for overdeletion and reinsertion repeatedly until the fix-point) transfer results back to the original ABox and yield an adapted abstraction. Since DRed operations on abstractions additionally require knowledge about the set of deletions, we extend the original definitions to bi-types and bi-abstractions.

**Definition 5 (Bi-Type).** Given two ABoxes \( \mathcal{A}_1, \mathcal{A}_2 \), the bi-type of an individual \( a \in \text{ind}(\mathcal{A}_1 \cup \mathcal{A}_2) \) w.r.t. \( (\mathcal{A}_1, \mathcal{A}_2) \) is a quadruple \( (\tau_{\mathcal{C}}^1(a), \tau_{\mathcal{R}}^1(a), \tau_{\mathcal{C}}^2(a), \tau_{\mathcal{R}}^2(a)) \), where \( (\tau_{\mathcal{C}}^1(a), \tau_{\mathcal{R}}^1(a)) \) is the combined type of a w.r.t. \( \mathcal{A}_1 \) and \( (\tau_{\mathcal{C}}^2(a), \tau_{\mathcal{R}}^2(a)) \) is the combined type of \( a \) w.r.t. \( \mathcal{A}_2 \).

**Definition 6 (Bi-Abstraction).** Given two ABoxes \( \mathcal{A}_1, \mathcal{A}_2 \) and a bi-type \( \tau = (\tau_{\mathcal{C}}^1, \tau_{\mathcal{R}}^1, \tau_{\mathcal{C}}^2, \tau_{\mathcal{R}}^2) \) w.r.t. \( (\mathcal{A}_1, \mathcal{A}_2) \), the bi-abstraction for \( \tau \) is an ABox \( B_\tau^1 \cup B_\tau^2 \), where \( B_\tau^1 = \{C(v) \mid C \in \tau_{\mathcal{C}}^1 \} \cup \{R(v, w) \mid R \in \tau_{\mathcal{R}}^1 \} \), and \( B_\tau^2 = \{C(v) \mid C \in \tau_{\mathcal{C}}^2 \} \cup \{R(v, w) \mid R \in \tau_{\mathcal{R}}^2 \} \), and \( v \) and \( w \) are distinguished abstract individuals for the bi-type \( \tau \). The
Algorithm 2: DRed(Ω, \( A^∞ \), \( A^− \))

In: \( \Omega = A \cup T \); ontology, \( A^∞ \): materialization, \( A^− \): deletions

Out: \( A^∞_{new} \): materialization of \( A \setminus A^− \) w.r.t. \( T \)

\( \triangleright \) overdeletion phase

1. \( D_{all} \leftarrow \) overdelete(\( \Omega, A^∞, A^− \))
2. \( A^∞_{new} \leftarrow A^∞ \setminus D_{all}, D_{new} \leftarrow D_{all} \cap (A \setminus A^−) \)
3. Apply the rules (Fig. 1) to \( A^∞_{new} \) w.r.t. \( T \) s.t. the conclusion is added to \( D_{new} \) and with the additional side condition that the conclusion occurs in \( D_{all} \)

\( \triangleright \) reinsertion phase

4. \( A^∞_{new} \leftarrow \) reinsert(\( \Omega, A^∞_{new}, D_{new} \))
5. return \( A^∞_{new} \)

bi-abstraction of \( (A_1, A_2) \) is \( B = B^1 \cup B^2 \) with

\[
B^1 = \bigcup_{a \in \text{ind}(A)} B^1_{\tau(a)} \quad \text{and} \quad B^2 = \bigcup_{a \in \text{ind}(A)} B^2_{\tau(a)},
\]

where \( \tau(a) \) is the bi-type for \( a \) w.r.t. \( (A_1, A_2) \) and \( B^1_{\tau(a)} \cup B^2_{\tau(a)} \) is the bi-abstraction for \( \tau(a) \).

The following example highlights how bi-abstractions also differentiate types based on their (over-)deleted assertions, while still aggregating ‘similar’ cases.

**Example 3.** Let \( A = \{A(a_1), A(a_2), A(a_3), r(a_1, b), r(a_2, b), r(a_3, b)\} \) and \( A^− = \{A(a_1), A(a_3)\} \), the combined type of \( a_1 \) and \( a_2 \) and \( a_3 \) w.r.t. \( A \) is \( \{\{\}, \{\}\} \). To distinguish \( a_1 \) and \( a_3 \) from \( a_2 \), we consider the bi-types w.r.t. \( (A \setminus A^−, A^−) \):

\[
\tau(a_1) = \tau(a_3) = (\emptyset, \{r\}, \{A\}, \emptyset), \tau(a_2) = (\{\}, \{r\}, \emptyset, \emptyset), \tau(b) = (\emptyset, \{r\}, \emptyset, \emptyset).
\]

Similar to the DRed algorithm, the Abstraction Refinement Delete and Rederive algorithm (ARDRed) consists of three consecutive phases: (1) compute the overdeletion using the function overdelete on abstractions followed by the refinement step until a fix-point is reached, (2) rederive deletions in the abstraction with an alternative derivation in one step, (3) reinsert facts with alternative derivations using Abstraction Refinement. For the reinsertion phase, we again restrict the derivation rules using the second part of bi-types, which represents the deleted assertions, as opposed to using the overdeletion set \( D_{all} \) in DRed.

We note two things about the overdeletion phase of Algorithm 3: First, the input ABox \( B^1 \cup B^2 \) for overdelete is not necessarily fully materialized, which can delay some derivations until the next loop round. Second, the function update uses the initial role assertions in \( A^∞ \), i.e. deletions are also propagated over deleted roles. The rederivation phase does not share parts with DRed (Algorithm 2), as we cannot restrict rederivations in a similar way, due to missing information about the concept deletions for role successors.

3.3 Correctness

It can easily be verified that the Abstraction Refinement aspects of the rederivation and reinsertion phases operate like the original Abstraction Refinement procedure and that the used rules are sound. Hence, soundness follows from the completeness of the overdeletion phase, i.e. we need to show that assertions no longer derivable from \( A \setminus A^− \) are not present when starting the rederivation phase. We make the notion of overdeletion precise using ABox justifications:

**Definition 7 (ABox Justification, Overdeletion).** Let \( \Omega = A \cup T \) be an ontology. For an axiom \( \alpha \) s.t. \( \Omega \models \alpha \), an ABox justification w.r.t. \( T \) is any set \( J \subseteq A \) s.t. \( J \cup T \models \alpha \) is minimal, if \( \forall J' \subseteq A \) s.t. \( J' \cup T \models \alpha \).

Let \( A^∞ \) be the materialization of \( A \) w.r.t. \( T \) and \( A^− \subseteq A \) deletions. Any superset of the set \( \{\alpha \in A^∞ \mid J \cup T \models \alpha \neq \emptyset \} \) for \( J \subseteq A \) a minimal ABox justification for \( \alpha \) w.r.t. \( T \) is an overdeletion of \( A^∞ \) w.r.t. \( A^− \).

For the overdeletion and reinsertion phases, we execute functions overdelete and reinsert over bi-abstractions and use homomorphisms to transfer the obtained results. Both functions construct closures via restricted derivations. For our argumentation, we consider two important properties induced by restricted derivations:

**Lemma 2 (Restricted Derivation Properties).** Let \( \Omega = A \cup T \) be an ontology and \( A_1, A_2 \) ABoxes with \( A_1 \subset A_2 \), \( \text{ind}(A_1) \subset \text{ind}(A) \) and \( B \) an ABox, s.t. there is a homomorphism \( h : \text{ind}(A) \rightarrow \text{ind}(B) \) from \( A \) to \( B \). Then the following properties hold for any assertion \( \alpha \):

\[
\begin{align*}
A \cup T \not\models_{A_1} \alpha & \implies A \cup T \not\models_{A_2} \alpha. \\
A \cup T \not\models_{A_1} \alpha & \implies B \cup T \models_{h(A_1)} h(\alpha).
\end{align*}
\]

The overdeletion/reinsertion phases terminate after reaching the fix-point, where no further information can be transferred using homomorphisms. We show that, in the fix-point, the closures constructed over bi-abstractions can be
lifted up to closures over the original ABoxes. For the inner overdeletion (line 5), we have $B^1 \cup B^2$ as the bi-abstraction w.r.t. $A^\infty \setminus D_{all} \cup B_2 \cup h \cup B_2 \alpha$. For the inner reinsertion, we have $B^1 \cup B^2$ as the bi-abstraction w.r.t. $A^\infty \setminus D_{all}$ (line 13). The inner reinsertion (line 14) constructs a closure for $B^1 \cup T \vdash B_1 \cup B_2 \alpha$. Note that, compared to the closure constructed by reinsert in the DRed algorithm, this is actually a restriction since (bi-)abstractions only contain concept assertions for individuals of the form $v_T$. Still, this restriction does not affect the final result. We can lift these closures up to $A^\infty \cup T \vdash D_{all} \alpha$ for the overdeletion and $A^\infty \cup T \vdash D_{all} \alpha$ for the reinsertion. Lemma 3 provides the proof for the overdeletion phase. The proof for the reinsertion phase is analogous.

**Lemma 3 (Overdeletion Fix-Point).** Let $O = A \cup T$ be a normalized Horn ALC\(H\)I ontology with materialization $A^\infty$ and deletions $A^- \subseteq A$. Further, let $D_{all}$ be s.t. $A^- \subseteq D_{all} \subseteq A^\infty$ and $B = B^1 \cup B^2$ the bi-abstraction w.r.t. $(A^\infty \setminus D_{all})$. 

For each individual $a \in \text{ind}(A)$ with bi-type $\tau = (\tau^\ell_a, \tau^r_a)$, each atomic concept $A \in \text{con}(O)$, and each role $R \in \text{rol}(O)$, if

1. $B^1 \cup B^2 \cup T \vdash A(v_T)$ implies $A(a) \in D_{all}$ and
2. $B^1 \cup B^2 \cup T \vdash B^2 A(w^R)$ and $R(a, b) \in A$ implies $A(b) \in D_{all}$,

then, for each assertion $\alpha$ s.t. $A^\infty \cup T \vdash D_{all} \alpha$, $\alpha \in D_{all}$.

**Sketch.** Extend $B = B^1 \cup B^2$ to $B' = B^1 \cup B'^2$ as follows: If $r(a, b) \in A^\infty \setminus D_{all}$, extend $B^1$ with $r(v_{T(a)}, v_{T(b)})$; if $r(a, b) \in A^-$, extend $B^2$ with $v_{T(a)} v_{T(b)}$. Note that there is a homomorphism $h$ from $A^\infty$ to $B'$, thus, by Lemma 2 (2), if $A^\infty \cup T \vdash D_{all} \alpha$, then $B'^1 \cup T \vdash h(D_{all}) \alpha$. We can then show that $h(D_{all}) \subseteq B'^2$, hence, by $B' \cup T \vdash B'^2 \alpha$.

We next sketch why $B'$ has the same atomic concept assertions as direct derivations under restriction $B'^2$ as $B$ under restriction $B^2$. Let $r(a, b) \in A^\infty$. By considering the construction of $B'$ from $B$, the rules in Figure 1, and the conditions of restricted derivations, we can show that introducing the new role assertion $v_{T(a)} v_{T(b)}$ does not result in new restricted entailments. Hence, according to condition 1, $A(a) \in D_{all}$.

**Lemma 4 (Soundness).** Let $O = A \cup T$ be a Horn ALC\(H\)I ontology, $A^\infty$ the materialization of $A$ w.r.t. $T$, $A^- \subseteq A$ a set of deletions, and $A^\infty_{new}$ the result of ARDRed$(O, A^\infty, A^-)$. If $\alpha \in A^\infty_{new}$, then $(A \setminus A^-) \cup T \vdash \alpha$.

**Sketch.** We mainly need to show that the set $D_{all}$ obtained as a closure according to Lemma 3 is complete and contains all assertions that are to be removed in the sense of Definition 7. To the contrary of what is to be shown, assume there is some $\alpha \notin D_{all}$ s.t. a minimal ABox justification $J$ for $\alpha$ contains an assertion from $A^-$. By Theorem 1, $\alpha$ can be derived via a number of rule applications from $J$ and, thus, there is a number of premises, of which at least one also has a minimal ABox justification $J' \subseteq J$ with $J' \cap A^- \neq \emptyset$. By doing so repeatedly, we eventually determine some assertion $\beta$ s.t. $A^\infty \cup T \vdash D_{all} \beta$, which is a contradiction. The ARDRed algorithm removes $D_{all}$ from $A^\infty$ and, as shown earlier, only sound assertions are reinserted.

We now show completeness using the previous results.

**Lemma 5 (Completeness).** Let $O = A \cup T$ be a Horn ALC\(H\)I ontology, $A^\infty$ the materialization of $A$ w.r.t. $T$, $A^- \subseteq A$ a set of deletions, and $A^\infty_{new}$ the result of ARDRed$(O, A^\infty, A^-)$. If $(A \setminus A^-) \cup T \vdash \alpha$, then $\alpha \in A^\infty_{new}$.

**Sketch.** First, we consider the inputs $A^\infty_{new}$ and $D_{new}$ as they are used in the closure construction of the reinsertion phase. By Lemma 4, $D_{all}$ contains the overdeletion of $A^\infty_{new}$ w.r.t. $A^-$. For $A^\infty_{new}$, we can then confirm that $A \setminus A^- \subseteq A^\infty_{new}$ using the definition of the algorithm. For $D_{new}$, we need to consider the effects of the rederviation phase. Using the rules in Figure 1 and the construction of the bi-abstraction $B_1 \cup B_2$, we can show that $D_{new}$ is determined s.t. it contains at least all assertions, which can directly be derived from $A^\infty \cup D_{all}$. It remains to show that under these conditions, constructing $A^\infty_{new}$ as a closure over $A^\infty_{new} \cup T \cup A^\infty' \cup D_{new}$ results in $A^\infty_{new}$ s.t. $(A \setminus A^-) \cup T \vdash \alpha$ implies $\alpha \in A^\infty_{new}$. Due to the initial construction of $A^\infty_{new}$, there must be some ‘initial culprit’ $\beta \notin A^\infty_{new}$ which can directly be derived from $A^\infty_{new}$ using a rule from Figure 1. We can then show that the conditions of the construction of $A^\infty_{new}$ are violated.

**Theorem 2 (Correctness).** Let $O = A \cup T$ be a Horn ALC\(H\)I ontology, $A^\infty$ the materialization of $A$ w.r.t. $T$, $A^- \subseteq A$ a set of deletions. ARDRed$(O, A^\infty, A^-)$ terminates and the returned $A^\infty_{new}$ is s.t. $\alpha \in A^\infty_{new}$ iff $(A \setminus A^-) \cup T \vdash \alpha$.

**Sketch.** The overdeletion and reinsertion loops of the algorithm terminate and, once an iteration cannot determine new assertions, we obtain the termination of the algorithm as a direct consequence of the already shown soundness (Lemma 4) and completeness (Lemma 5) results. In particular, termination occurs in the worst case, when $D_{all} = A^\infty$ (overdeletion) and $A^\infty_{new} = A^\infty$ (reinsertion).

### 4 Implementation and Evaluation

For the evaluation, we focus on directly comparing the algorithms, i.e. classical DRed and ARDRed. To do so, we have implemented both from scratch in Java including support for additions. The code and the experimental data are publicly available [Glimm and Brenner, 2018]. As basis for the test cases, we used the UOBM benchmark [Ma et al., 2006], for which we generated instances of 10, 50 and 100 universities (denoted UOBM10, UOBM50, and UOBM100, respectively) and the well-known NPD and Reactome ontologies. All ontologies were preprocessed by dropping non-Horn ALC\(H\)I axioms and by normalizing the remaining axioms. Table 1 shows the number of TBox and ABox axioms of the preprocessed ontologies.
An incremental test consists of a TBox \( \mathcal{T} \) and a base ABox \( \mathcal{A} \) from which we obtain an initial ABox \( \mathcal{A}_0 \) and 100 change ABoxes containing assertions to be either added or deleted, depending on the test scenario. Each change ABox contains all assertions for 1% of the individuals from \( \mathcal{A} \).\(^1\) We evaluate three scenarios: (1) add-only starts with an almost empty \( \mathcal{A}_0 \) and step by step adds the change ABoxes; (2) remove-only starts with \( \mathcal{A}_0 = \mathcal{A} \) and then removes the change ABoxes; (3) add-remove alternates between additions and removals, starting with two initial addition steps to avoid adding/removing the same individuals over and over again. All tests were executed on a server with two Intel hexa core processors at 2.60GHz (without using parallelization) and each test execution was assigned 200 GB of the overall 500 GB RAM to allow for keeping all test data in memory to avoid I/O impact. The number of 100 change ABoxes is large enough to even out JVM behavior (e.g. code optimization at runtime).

Results are shown in Table 1 and visualized in Figure 2. It can be seen that our approach outperforms the classical DRed implementation in almost all cases. We can also observe the typical Abstraction Refinement behavior, in that the advantage of ARDRed over DRed increases with the ABox size: For UOBM\textsubscript{10}, DRed still outperforms ARDRed in the add-only and remove-only cases. Figure 2c shows initial reasoning times stacked under the streaming times to display accumulated results. When running an incremental reasoning system for a longer time period, initial reasoning times eventually become irrelevant. Considering this argument, ARDRed behaves slightly worse than DRed for NPD and Reactome in the remove-only case. Our analysis shows that this is due to the unoptimized bi-types, which currently require individuals to start with the same materialization and to receive the same deletions, to be classified into the same type and we presume that this restriction is too strong for practical ontologies. This does not seem to be a problem for the UOBM ontologies, which we attribute to the synthetic data generation. The improvement achieved by ARDRed over DRed is less noticeable than that for non-incremental materialization by Abstraction Refinement. We attribute this to a generally increased overhead of the abstraction construction and refinement aspects in the incremental setting, which is confirmed by the fact that the initial materialization times for \( \mathcal{A}_0 \) in the remove-only setting are comparable to those of the original Abstraction Refinement approach. The runtimes of the add-remove case are generally lower than the runtimes of the add-only case, which in return are lower than those of the remove-only case. The first can be explained by the general number of assertions, which is lower in the add-Remove case (as we keep adding and removing the same number of individuals). The second result is also to be expected, as deletion of assertions is far more expensive than adding assertions.

\(^1\)We exclude individuals which are used like nominals (implicitly belonging to the terminological part), e.g. the individual realizing the SwimmingClass concept in UOBM. Such individuals make up far less than 1% of the total individuals.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline
\textbf{Ontology} & \textbf{\(|\mathcal{T}|\)} & \textbf{\(|\mathcal{A}|\)} & \multicolumn{2}{c|}{\textbf{add-only}} & \multicolumn{2}{c|}{\textbf{add/\textit{remove}}} & \multicolumn{2}{c|}{\textbf{remove-only}} \\
\hline
\textbf{UOBM\textsubscript{10}} & 498 & 1926897 & ARDRed stream & 19.03 & 8.44 & ARDRed stream & 114.84 & \textit{init} stream & 7.22 \\
\textbf{UOBM\textsubscript{50}} & 498 & 9751681 & DRed stream & 87.88 & 32.22 & DRed stream & 9751681 & \textit{init} stream & 42.22 \\
\textbf{UOBM\textsubscript{100}} & 498 & 19571755 & ARDRed stream & 176.26 & 59.49 & ARDRed stream & 498 & \textit{init} stream & 96.56 \\
\textbf{NPD} & 1241 & 911517 & DRed stream & 61.5 & 3.39 & DRed stream & 287.41 & \textit{init} stream & 24.83 \\
\textbf{Reactome} & 597 & 7087410 & ARDRed stream & 67.43 & 3.06 & ARDRed stream & 318.5 & \textit{init} stream & 25.83 \\
\hline
\end{tabular}
\caption{Test ontologies with the number of axioms (\(|\mathcal{T}|\)) and assertions (\(|\mathcal{A}|\)) for their Horn ALCHI subsets with the stream materialization times in seconds. Materialization time for \( \mathcal{A}_0 \) (init) is only given for the remove-only case (the times for the other cases are < 200 ms).}
\end{table}

5 Conclusion and Future Work

The introduced novel materialization maintenance algorithm for Horn ALCHI ontologies combines DRed and Abstraction Refinement. Benefits of this approach lie in the summarization of similar deletion and reasoning tasks, paving the road for efficient maintenance of materializations of large ABoxes.

For the implementation, further optimizations for the Abstraction Refinement part (e.g. caching of types, further summarization of types in the overdeletion) and the DRed part (e.g. integrating the Backward/Forward optimization [Motik et al., 2015]) are paths to be explored.
Acknowledgements

This work was done within the project “Reasoning over Large Amounts of Data in Ontologies via Abstraction and Refinement” (GL 779/3-2) and the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References


