Beyond Asserted Axioms: Fine-Grain Justifications for OWL-DL Entailments

Aditya Kalyanpur[1] and Bijan Parsia[1] and Bernardo Cuenca Grau[2]
University of Maryland, USA[1], University of Manchester, UK[2]
aditya@cs.umd.edu, bparsia@isr.umd.edu, bcg@cs.man.ac.uk

1 Motivation

The Ontology Engineering community widely agrees on the importance of helping the user understand the output of a DL reasoner. The most recent approaches to the problem [4] [3] are based on the notion of a *Minimal Unsatisfiability Preserving Sub-TBoxes* (MUPS). Roughly, a MUPS for an atomic concept A is a minimal fragment $\mathcal{T}' \subseteq \mathcal{T}$ of a TBox $\mathcal{T}$ in which A is unsatisfiable.

For example, given the TBox:

1: $A \sqcup C \sqsubseteq B \sqcap \exists R.D \sqcap E \sqcap \neg B$, 2: $C \sqsubseteq F \sqcup \geq 1.R$
3: $A \sqsubseteq E \sqcap \forall R.\neg D$, 4: $B \sqsubseteq \neg D \sqcup E$

where the concept A is unsatisfiable, the TBox $\mathcal{T}' = \{1\}$ is a MUPS for A in $\mathcal{T}$.

In our earlier work [2], we extended the notion of a MUPS to arbitrary entailments in $\mathcal{SHOIN}$ and devised a non-standard inference service, *axiom pinpointing*, which, given $\mathcal{T}$ any of its logical consequences α, determines the minimal fragments of $\mathcal{T}$ in which α holds.

However, these services, though useful for debugging and explanation purposes, suffer from a fundamental *granularity* problem: since they work at the asserted axiom level, they fail to determine which *parts* of the asserted axioms are *irrelevant* for the particular entailment under consideration to hold. For instance, in our example, the conjunct E in axiom 1 is irrelevant for the unsatisfiability of A. Moreover, additional parts of axioms that could contribute to the entailment are *masked*, e.g., the axioms 1, 3 can be broken down into $A \sqsubseteq \exists R.D$ and $A \sqsubseteq \forall R.\neg D$ which also entail the unsatisfiability of A, however, this condition cannot be captured.

In this paper, we aim at extending the axiom pinpointing service to capture *precise justifications*, which are at a finer granularity level than the original asserted axioms. In this context, we provide a formal notion of precise justification and propose a decision procedure for the problem. We discuss implementation details of the service

and show that it is feasible in practice. Our results are applicable to $\mathcal{SHOIN}$ and hence to OWL-DL.

2 Precise Justifications

Since we aim at identifying relevant parts of axioms, we define a function that splits the axioms in a TBox $\mathcal{T}$ into "smaller" axioms to obtain an equivalent TBox $\mathcal{T}_s$ that contains as many axioms as possible.

Definition 1 ***(split_KB)*** *Given a $\mathcal{SHOIN}$ concept C in negation normal form (NNF), the function $split(C)$ returns a set of concepts, inductively defined as follows:*

$split(C) \leftarrow C$, *if C is $A, \neg A, \geq n.R, \leq n.R, = n.R, \top, \bot$ or a nominal node $\{o\}$*
$split(C \sqcap D) \leftarrow split(C) \cup split(D)$
$split(C \sqcup D) \leftarrow (split(C) \otimes_{\sqcup} split(D))$
$split(\forall R.C) \leftarrow \forall R. \prod(split(C))$
$split(\exists R.C')$ – *we get two cases depending on C'*
if C' is of the form $C \sqcap D$,
$split(\exists R.C') \leftarrow \exists R.E \cup split(\neg E \sqcup C) \cup split(\neg E \sqcup D) \cup split((\neg C \sqcup \neg D) \sqcup E)$, *where E is a fresh concept*
else,
$split(\exists R.C') \leftarrow \exists R. \prod split(C')$

Given a KB $= \{\alpha_1, \alpha_2, ...\alpha_n\}$, where each axiom α_i is of the form $C_i \sqsubseteq D_i$, let C_α be the concept representing α, i.e., $\neg C_i \sqcup D_i$, converted to NNF. Then, $T_s = split\_KB() = \sum \top \sqsubseteq C_s$ for each concept $C_s \in split(\neg C_1 \sqcup D_1) \cup split(\neg C_2 \sqcup D_2)..split(\neg C_n \sqcup D_n)$.

The idea of the above function is to rewrite the axioms in in a convenient normal form and split across conjunctions in the normalized version, e.g., rewriting $A \sqsubseteq C \sqcap D$ as $A \sqsubseteq C, A \sqsubseteq D$.[1] In some cases, we are forced to introduce new concept names, only for the purpose of splitting axioms into smaller sizes (which prevents any arbitrary introduction of new concepts); for example, since the axiom $A \sqsubseteq \exists R.(C \sqcap D)$ is not equivalent to $A \sqsubseteq \exists R.C$, $A \sqsubseteq \exists R.D$, we introduce a new concept name, say E, and transform the original axiom into the following set of "smaller" axioms: $A \sqsubseteq \exists R.E$, $E \sqsubseteq C$, $E \sqsubseteq D$, $C \sqcap D \sqsubseteq E$.

Table 1 shows an algorithm to split a TBox based on the above definition. In this algorithm, we also keep track of the correspondence between the new axioms and the axioms in $\mathcal{T}$ by using a function σ.

[1]Note that the $\otimes_{\sqcup}$ operator generates a cross-product of disjunctions, e.g, $split((A \sqcap B) \sqcup (C \sqcap D)) = \{(A \sqcup C), (A \sqcup D), (B \sqcup C), (B \sqcup D)\}$; and the $\prod$ operator applies the set of concepts to the preceding universal or existential operator, e.g., $split(\forall R.(B \sqcap (C \sqcup D))) = \{\forall R.B, \forall R.(C \sqcup D)\}$.

Algorithm: Split KB
Input: TBox $\mathcal{T}$
Output: TBox $\mathcal{T}_s$, Axiom Correspondence Function σ

$\mathcal{T}_s \leftarrow \emptyset$
initialize axiom correspondence function σ
initialize substitution $cache$
for each subclass axiom $\alpha \in \mathcal{T}$
    from α := $C \sqsubseteq D$ generate C_α := $\neg C \sqcup D$
    normalize C_α to NNF (pushing negation inwards)
    $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \{\top \sqsubseteq C_\alpha\}$
    $\sigma(\{\top \sqsubseteq C_\alpha\}) \leftarrow \alpha$
while there exists an axiom $\{\top \sqsubseteq C_\alpha\} \in \mathcal{T}_s$ s.t. **can-split**$(C_\alpha) \neq \emptyset$,
    $\mathcal{T}_s \leftarrow \mathcal{T}_s - \{\top \sqsubseteq C_\alpha\}$
    $\langle T, v, \pi, A, B\rangle \leftarrow$ **can-split**(C_α)
    if $L_t(\langle \prec_t (v), v\rangle) \neq \exists R$, **then**
        $C_A \leftarrow C_\alpha[A]_\pi$; $\sigma(C_A) \leftarrow \sigma(C_A) \cup \sigma(C_\alpha)$; $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \{\top \sqsubseteq C_A\}$
        $C_B \leftarrow C_\alpha[B]_\pi$; $\sigma(C_B) \leftarrow \sigma(C_B) \cup \sigma(C_\alpha)$; $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \{\top \sqsubseteq C_B\}$
    else
      if $cache(A \sqcap B) = \emptyset$, **then**
          let E be a new concept not defined in $\mathcal{T}_s$
          $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \{E \sqsubseteq A, E \sqsubseteq B, A \sqcap B \sqsubseteq E\}$
          $cache(A \sqcap B) \leftarrow E$
      else $E \leftarrow cache(A \sqcap B)$
      $C_E \leftarrow C_\alpha[E]_\pi$; $\sigma(C_E) \leftarrow \sigma(C_\alpha)$; $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \{\top \sqsubseteq C_E\}$

$subroutine$: **can-split**(C_α)
let $T = (W_t, \prec_t, L_t)$ be the parse tree for the disjunction C_α
if there exists a node $v \in W_t$ in T s.t. $L_t(v) = \sqcap$ with children A, B
    let π be the position of node v
    return $\langle T, v, \pi, A, B\rangle$
else return $\emptyset$

Table 1: Splitting a TBox

A *justification* is a generalization of the notion of a MUPS for arbitrary entailments:

Definition 2 ***(Justification)*** *Let $\mathcal{T} \models \alpha$ where α is an axiom. A fragment $\mathcal{T}' \subseteq \mathcal{T}$ is a justification for α in $\mathcal{T}$ if $\mathcal{T}' \models \alpha$, and $\mathcal{T}'' \not\models \alpha$ for every $\mathcal{T}'' \subset \mathcal{T}'$.*

We denote by $\mathsf{JUST}(\alpha, \mathcal{T})$ the set of all justifications for α in $\mathcal{T}$.

Finer-grained justifications can be defined as follows:

Definition 3 ***(Precise Justification)***
*Let $\mathcal{T} \models \alpha$. A TBox $\mathcal{T}'$ is a **precise justification** for α in $\mathcal{T}$ if $T' \in \mathsf{JUST}(\alpha, \mathsf{split\_TBox}(\mathcal{T}))$.*

We denote by $\mathsf{JUST}_\mathsf{p}(\alpha, \mathcal{T})$ the set of all precise justifications for α in $\mathcal{T}$.

In $\mathcal{SHOIN}$, entailment can be reduced to concept unsatisfiability. As a consequence, it is easy to show that the problem of finding all the justifications for an

unsatisfiable concept and the problem of finding all the justifications for a given entailment can be reduced to each other. In what follows, we shall restrict our attention, without loss of generality, to the problem of finding all the precise justifications for an unsatisfiable concept w.r.t to a $\mathcal{SHOIN}$ TBox.

3 Finding Precise Justifications

The problem of finding all precise justifications for an entailment α of a TBox $\mathcal{T}$ now reduces to the problem of finding all justifications for α in the $split\_TBox(\mathcal{T})$. Thus, we can use the algorithm listed in Table 1 to split a TBox, and then apply any decision procedure to find all justifications for the entailment in the split version of the TBox.

We briefly sketch the algorithm we have developed in our earlier work to find all justifications for an arbitrary entailment in a TBox. For details, we refer the reader to the technical report [2].

The first part of the algorithm is based on the decision procedure for concept satisfiability in $\mathcal{SHOIN}$ [1], and it finds any one arbitrary justification for the entailment. It keeps track of the axioms from the TBox responsible for each change in the completion graph, namely, the addition of a particular concept (or role) to the label of a specific node (or edge), the addition of an equality (merge) or inequality relation between nodes, or the detection of a contradiction (clash) in the label of a node.

In order to ensure correctness, our algorithm imposes an ordering among the deterministic rules: the $unfolding$ and CE rules are only applied when *no other* deterministic rule is applicable. The rationale for this strategy relies on the definition of justification that establishes minimality w.r.t. the number of axioms considered; the new rules cause new axioms to be considered in the tracing process and additional axioms should only be considered if strictly necessary.

The algorithm works on a tree $\mathbf{T}$ of completion graphs. that is incrementally built using the set of available *expansion rules*, and the traces for the events triggered by the rules are updated on the fly. For a full specification of the expansion rules, we refer the reader to [2]. The application of non-deterministic rules results in the addition of new completion graphs as leaves of $\mathbf{T}$, one for each different non-deterministic choice. However, since C is unsatisfiable w.r.t. $\mathcal{T}$, the algorithm detects a clash in each of the leaves. Upon termination, the trace of the detected clash (or clashes) in the leaves of $\mathbf{T}$ yields a justification.

Once a single justification has been found, we find the remaining justifications by employing a variation of the classical Hitting Set Tree (HST) algorithm.

Given a collection S of conflict sets, Reiter's algorithm constructs a labeled tree called *Hitting Set Tree* (HST). Nodes in an HST are labeled with a set $s \in S$ and: **1)** if $H(v)$ is the set of edge labels on the path from the root to the node v, then $\mathcal{L}(v) \cap H(v) = \emptyset$; **2)** for each $\sigma \in \mathcal{L}(v)$, v has a successor w and $\mathcal{L}(\langle v, w \rangle) = \sigma$; **3)** if $\mathcal{L}(v) = \emptyset$, then $H(v)$ is a hitting set for S.

In our approach, we form an HST where nodes correspond to single justification sets (computed on the fly), edges correspond to the removal of axioms from the KB, and in building the tree to compute all the minimal hitting sets, we in turn obtain all the justification sets as well. The correctness and completeness of this approach is provided in [2].

3.1 Optional Post-Processing

Having derived the justifications for the entailment $A \equiv \bot$ in the maximally split version of KB $\mathcal{T}$, we remove any term in a justification set that was introduced in the KB in step 1 using the procedure described in Table 2. The output of this stage gives us precise justifications for the entailment in terms of sub-axioms.

Algorithm: Remove New Terms
Input: Collection of Axiom Sets $\mathcal{J}$, Original KB $\mathcal{T}$
Output: $\mathcal{J}$
for each axiom set $j \in J$ **do**,
while there exists a term $C' \in \mathsf{vocab(j)}$ s.t. $C' \notin \mathsf{vocab}(\mathcal{T})$, **do**
$S \leftarrow \emptyset$
for each axiom $C' \sqsubseteq C_i \in j$, **do**
$S \leftarrow S \cup C_i$
if $S \neq \emptyset$, **then**
$C \leftarrow C_1 \sqcap C_2 \sqcap ..C_n$ (for all $C_i \in S$)
else
$C \leftarrow \top$
substitute C' with C in j

Table 2: Post-Processing

3.2 Example

We now present a detailed example to demonstrate how the algorithm finds precise justifications correctly.

Consider a KB $\mathcal{T}$ composed of the following axioms:

1. $A \sqcup B \sqsubseteq \exists R.(C \sqcap \neg C) \sqcap D \sqcap E$
2. $A \sqsubseteq \neg D \sqcap B \sqcap F \sqcap D \sqcap \forall R.\bot$
3. $E \sqsubseteq \forall R.(\neg C \sqcap G)$

Note, $\mathsf{vocab}(\mathcal{T}) = \{A, B, C, D, E, F, R\}$, and the concept A is unsatisfiable w.r.t $\mathcal{T}$.

Given $A, \mathcal{T}$ as input, the algorithm proceeds as follows:

Step 1: After pre-processing, we obtain a maximally split equivalent KB $\mathcal{T}_s$:

$T_s = \{A \sqsubseteq \exists R.H^1; B \sqsubseteq \exists R.H^1; A \sqsubseteq D^{1,2}; B \sqsubseteq D^1; A \sqsubseteq E^1; B \sqsubseteq E^1; H \sqsubseteq C^1; H \sqsubseteq \neg C^1; A \sqsubseteq \neg D^2; A \sqsubseteq B^2; A \sqsubseteq F^2; A \sqsubseteq \forall R.\bot^2; E \sqsubseteq \forall R.\neg C^3; E \sqsubseteq \forall R.G^3\}$.

The superscript of each axiom in $\mathcal{T}_s$ denotes the corresponding axiom in $\mathcal{T}$ that it is obtained from. This correspondence is captured by the function σ in the Split-KB algorithm (see Table 1). Notice that the superscript of the axiom $A \sqsubseteq D$ in $\mathcal{T}_s$ is the set $\{1, 2\}$ since it can be obtained from two separate axioms in $\mathcal{T}$. Also, we have introduced a new concept H in the split KB, which is used to split the concept $\exists R.(C \sqcap \neg C)$ in axiom 1.

Step 2: Now, we obtain the justifications for the unsatisfiability of A w.r.t $\mathcal{T}_s$. This gives us the following axiom sets J:

$J = \{\{A \sqsubseteq \exists R.H^1, H \sqsubseteq C^1, H \sqsubseteq \neg C^1\}; \{A \sqsubseteq D^{1,2}, A \sqsubseteq \neg D^2\}; \{A \sqsubseteq \exists R.H^1, H \sqsubseteq C^1, A \sqsubseteq E^1, E \sqsubseteq \forall R.\neg C^3\}; \{A \sqsubseteq \exists R.H^1, A \sqsubseteq \forall R.\bot^2\}\}$

Step 3: Finally, we remove the concept H introduced in $\mathcal{T}_s$ from the justification sets in J to get:

$\mathcal{T}' = \{\{A \sqsubseteq \exists R.(C \sqcap \neg C)^1\}; \{A \sqsubseteq D^1, A \sqsubseteq \neg D^2\}; \{A \sqsubseteq D^2, A \sqsubseteq \neg D^2\}; \{A \sqsubseteq \exists R.C^1, A \sqsubseteq E^1, E \sqsubseteq \forall R.\neg C^3\}; \{A \sqsubseteq \exists R.\top^1, A \sqsubseteq \forall R.\bot^2\}\}$

As can be seen, the final output is the complete set of precise justifications for $A \equiv \bot$ in $\mathcal{T}$.

4 Implementation Issues

We have shown in [3], [2] that the axiom pinpointing service used to find all justifications for an entailment of a $\mathcal{SHOIN}$ KB performs reasonably well on a lot of real-world OWL-DL ontologies.

The main additional overhead incurred for capturing precise justifications is due to the pre-processing phase where we split the KB. The concern here, from a reasoning point of view, is the introduction of GCIs during the splitting process, e.g. $A \equiv B \sqcap C$ is replaced by (among other things) $B \sqcap C \sqsubseteq A$. Even though these GCIs are absorbed, they still manifest as disjunctions and hence adversely affect the tableau reasoning process. [2]

Alternately, a more optimal implementation is the following: instead of splitting the entire KB beforehand, we can perform a *lazy splitting* of certain specific axioms (on the fly) in order to improve the performance of the algorithm. The modified algorithm with lazy splitting becomes:

- Given A unsatisfiable w.r.t $\mathcal{T}$, find a single justification set, $J \in \mathsf{JUST}(\mathsf{A} \equiv \bot, \mathcal{T})$
- Split axioms in J to give J_s. Prune J_s to arrive at a minimal precise justification set J_p

[2] In this case, it is possible to introduce a new absorption rule in the reasoner which would transform axioms of the form $B \sqsubseteq A, C \sqsubseteq A, B \sqcap C \sqsubseteq A$ to $A \equiv B \sqcap C$ internally. This obviously seems odd given that we split the KB initially and re-combine axioms back in the reasoner, but note that the goal here is finding precise justifications.

- Replace J by J_s in $\mathcal{T}$.
- Form Reiter's HST using J_p as a node, with each outgoing edge being an axiom $\alpha \in J_p$ that is removed from $\mathcal{T}$

The advantage of this approach is that it only splits axioms in the intermediate justification sets in order to arrive at precise justifications, and re-inserts split axioms back into the KB dynamically. For details of the procedure, see [2]

5 Conclusion and Future Work

We have implemented the axiom pinpointing service in the OWL-DL reasoner, Pellet, and exposed it in the OWL Ontology Editor, Swoop. We are in the process of extending it to capture precise justifications.

From a UI perspective, maintaining the correspondence between the subaxioms in the precise justification and the original asserted axioms is useful, as it allows us to strike out irrelevant parts of the axioms directly, which is helpful to understand the entailment better. (see Figure 5).

As future work, we plan to optimize the implementation and do a thorough performance and usability evaluation.

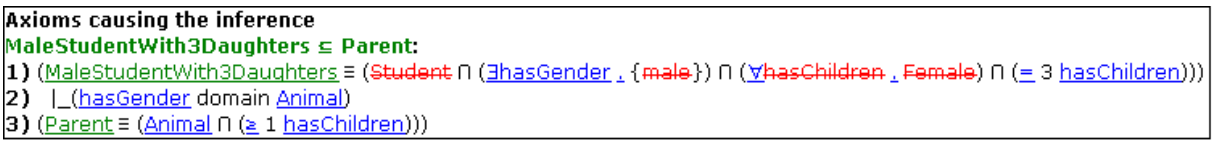

Figure 1: Justification for MaleStudentWith3Daughters $\sqsubseteq$ Person in the Koala OWL Ontology

References

[1] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for SHOIQ. In *Proc. of IJCAI 2005*, 2005.

[2] Aditya Kalyanpur, Bijan Parsia, Bernardo Cuenca, and Evren Sirin. Axiom pinpointing: Finding (precise) justifications for arbitrary entailments in OWL-DL, 2006. Available at http://www.mindswap.org/papers/2006/AxiomPinpointing.pdf.

[3] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. *J. of Web Semantics, Vol 3, Issue 4*, 2005.

[4] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proc. of IJCAI, 2003*, 2003.