

Explaining Entailments and Patching Modelling Flaws

for Ontology Authoring

Thorsten Liebig, Stephan Scheele

Ontology authoring is a sophisticated task and requires domain as well as some amount of background knowledge in formal logic. In fact, it is not only novice users that are commonly faced with comprehension problems with respect to the influence of complex or nested ontological axioms on reasoning. We provide practical insights into the development of tableau-based methods for explaining the key inference services, namely unsatisfiability, subsumption, and non-subsumption as well as techniques for patching ontologies in order to establish a user desired entailment.

1 Introduction and Approach

Ontologies are key to *semantic technologies* which are widely seen as a new paradigm of intelligent information processing. However, creating consistent ontologies which actually express what the knowledge engineer intended to model has shown to be a non-trivial task. For example, implicit modelling conflicts or a misunderstanding of the inference algorithm were responsible for a significant amount of system failures in an analysis of different efforts of formalizing knowledge by KR experts [4]. In order to avoid modelling problems more effort should be put into the development of interactive tools for building, maintaining and evaluating ontologies. Consequently, an appropriate ontology authoring tool has to support the user in grasping the logical consequences of an axiom. This requires the availability of non-standard inference services such as explaining or patching. Whereas explaining has become an active research area, patching still has not been seriously addressed yet. Our work implements the idea of generating human comprehensible explanations for certain entailments by compiling parametrised text patterns that are triggered by a proof algorithm [8]. We use a two-phase approach, which first builds the proof and collects information about those axioms and proof steps responsible for the entailment. In a second step the collected proof structures are post-processed to generate quasi-natural language explanation steps on a level which is traceable for the end user. Our method is capable of explaining ontologies with an expressivity of the Description Logic (DL) *SHIN* [1] which covers a significant fraction of the Web Ontology Language (OWL) and most of the ontologies found on the Web or in semantic applications today. Concept descriptions in *SHIN* are formed according to the following rules:

$$C, D \rightarrow A \mid \top \mid \perp \mid \neg C \mid C \sqcap D \\ \mid C \sqcup D \mid \exists R.C \mid \forall R.C \mid (\{\leq, \geq, =\}n R)$$

DL semantics are defined by set-theoretic interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non-empty set $\Delta^{\mathcal{I}}$ of individuals and an interpretation function $\cdot^{\mathcal{I}}$ mapping each concept A to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and every property to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. \mathcal{I} is inductively extended to arbitrary formulae (e.g. see [1]).

Tableau-based Explaining. Tableau algorithms [1] are based on the refutation principle, whose objective is to prove a formula by showing that its negation cannot be satisfied. The reasoning procedure builds a so-called completion graph where each node corresponds to an individual that is labelled with the set of formulae $\mathcal{L}(x)$ it has to satisfy, i.e. if $\mathcal{L}(x) = \{A\}$ then $x \in A^{\mathcal{I}}$. Each edge (x, y) in the graph represents a pair occurring in the interpretation of the property, i.e. if $\mathcal{L}(x, y) = \{R\}$ then $(x, y) \in R^{\mathcal{I}}$, and is labeled with a set of property names. The completion graph is composed starting from the root node by applying specific expansion rules extending the graph successively. The algorithm stops processing a node if no more rules are applicable or if an obvious contradiction (so-called clash) [1, p.80] has been found in a node, e.g. $\{A, \neg A\} \subseteq \mathcal{L}(x)$ for some node x . The algorithm terminates if all alternative branches either lead to a clash or cannot be further expanded. In the case of a closed tableau where all branches do clash, the unsatisfiability of the root-node has been proven.

2 Tableau Inference Illustration

We assume that humans typically do not conclude in a refutation style, furthermore we argue that syntactical optimization procedures used in tableau algorithms of well-known reasoning systems such as RacerPro are counterproductive because they obscure the structure of the original query.

Tagging. We address the problem of structure-destroying transformations by applying a so-called tagging [2] technique which labels the subsumer of a subsumption query with a special flag (\dagger in the following). This allows to distinguish between terms from the subsumee (lhs) and subsumer (rhs) in order to reconstruct the original inference problem out of the corresponding refutation problem at any stage during tableaux processing. E.g. in a query $A \sqsubseteq C$ the right-hand side is tagged in the corresponding refutation problem, i.e. $A \sqcap \neg C^\dagger$.

Drill-down Explanations. Our approach assumes that humans usually drill down into a problem by reducing it into