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. This work is driven by the question whether inference illustration techniques are suitable to support ontology engineers in understanding inferences as well as in identifying and patching certain modelling flaws. 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 by suggesting modelling changes 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. Therefore, the availability of high quality ontologies will play an important role in a market which is expected to grow at the rate of 40 % per year up to $52 billion in 2010 [5]. 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 modeling 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 [9]. Inexperienced users may even mistrust certain inference results because of unintended outcomes. In fact, many consequences are not easily traceable or counterintuitive mainly because they often depend on logical interrelations with other definitions or nested definitions, which themselves may depend on other consequences. In order to avoid those modeling problems the authors of the mentioned analysis suggest to put more effort into the development of interactive tools for building, maintaining and evaluating ontologies.

Consequently, an appropriate ontology authoring tool not only has to provide an effective browsing interface or intuitive editing capabilities, but also has to support the user in grasping the logical consequences of the axioms. Furthermore, an important but largely neglected kind of service is to explain why an unexpected entailment holds or a desired conclusion is missing. In the latter case the service should provide concrete hints which help to improve the users modeling decisions so that they logically define what she/he originally intended to express. These two tasks, namely explaining as well as patching, are non-standard reasoning services which are beyond the scope of typical reasoning systems. Whereas explaining has become an active research area in ontology reasoning recently, patching still hasn't been seriously addressed yet.

This paper provides an overview about the conceptual approach as well as implementation issues of our explanation and patching component. These services are especially useful during ontology building or maintenance and have been integrated into our ontology authoring tool OntoTrack [15].

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. This results in on-demand quasi-natural language explanations for the key inference services for ontology authoring, namely unsatisfiability, subsumption, and non-subsumption. Hereby we use a two-phase approach, which first builds the proof and collects information about those axioms and proof steps responsible for the entailment, and then traverses the collected proof structures in order to generate quasi-natural language explanation steps on a level which is traceable for the end user. The system we implemented is based on an extended tableau algorithm for explaining as originally suggested in [3]. It is capable of explaining ontologies with an expressivity of the Description Logic (DL) $\mathcal{SHN}$ [1] whereas our approach theoretically could handle $\mathcal{SHIN}$. $\mathcal{SHN}$ covers a significant fraction of the Web Ontology Language and most of the ontologies found on the Web or in semantic applications today.

2 Preliminaries

2.1 OWL – Syntax and Semantics

The Web Ontology Language (OWL) is the W3C recommended formal language for the representation and interchange of ontological knowledge on the Web. It is based on Description Logics and layered on top of Web standards such as the Resource Description Framework (RDF – an abstract data model) and RDF Schema (RDF incl. a schema extension). OWL comes in three increasingly expressive languages known as OWL Lite, OWL DL and OWL Full which differ in logical expressivity [2].

The most expressive as well as decidable OWL language fragment is OWL DL. It roughly corresponds to the Description Logic $\mathcal{SHOIN}$ which is based on the following denumerable sets: the set of role names N_R, the set of concept names N_C and the set N_I of individual names. Concept descriptions in $\mathcal{SHOIN}$ are formed according to the following rules:

$$\begin{aligned} C, D \quad \rightarrow \quad & A \mid \top \mid \bot \mid \neg C \mid \{o\} \mid C \sqcap D \\ & \mid C \sqcup D \mid \exists R.C \mid \forall R.C \mid (\{\leq, \geq, =\} n\ R) \end{aligned}$$

where $A \in N_C$ (A denoting the atomic concept), $R \in N_R$, $o \in N_I$ and $n \in \mathbb{N}_0$. An interpretation $\mathcal{I}$ of $\mathcal{SHOIN}$ is a structure $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non-empty set $\Delta^{\mathcal{I}}$ of individuals (the

domain) 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 $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to concept descriptions by the following inductive definition:

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
\{o\}^{\mathcal{I}} &= \{o^{\mathcal{I}}\} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \left\{a \in \Delta^{\mathcal{I}} \mid \exists b.\ (a,b) \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\right\} \\
(\forall R.C)^{\mathcal{I}} &= \left\{a \in \Delta^{\mathcal{I}} \mid \forall b.\ (a,b) \in R^{\mathcal{I}} \Rightarrow b \in C^{\mathcal{I}}\right\} \\
(\geq n\, R)^{\mathcal{I}} &= \left\{a \in \Delta^{\mathcal{I}} \;\middle|\; |\left\{b \mid (a,b) \in R^{\mathcal{I}}\right\}| \geq n\right\} \\
(\leq n\, R)^{\mathcal{I}} &= \left\{a \in \Delta^{\mathcal{I}} \;\middle|\; |\left\{b \mid (a,b) \in R^{\mathcal{I}}\right\}| \leq n\right\}
\end{aligned}$$

In OWL DL properties can be declared as symmetric, transitive, functional, or as inverse of another property. Furthermore the domain and range of a property can be globally restricted.

The key inference service of ontological reasoners is subsumption. A class C is said to subsume a class D (written $D \sqsubseteq C$), iff the set-theoretic interpretation of D is a subset of $C^{\mathcal{I}}$ ($D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$), or logically speaking iff the left hand side (lhs) implies the right hand side (rhs). With respect to class centered formalisms the subsumption relationship corresponds to the sub-class relation, i.e. if $D \sqsubseteq C$ then D is a subclass of C. Furthermore, the inference services unsatisfiability and non-subsumption are reducible to the subsumption problem [1].

2.2 Tableau Algorithms

Our approach relies on a modified tableau proof algorithm. Due to the lack of space we will only sketch the procedure of DL-tableau algorithms, for a detailed explanation see [1]. Tableau algorithms are based on the refutation principle, whose objective is to prove a formula by showing that its negation cannot be satisfied. This is done in a constructive manner which tries to create a model of the negated formula by applying tableau expansion rules until a contradiction or model is found. The initial expressions build the root node of a so-called *completion graph*. Each node x in the graph represents an individual, labeled with the set of concepts $\mathcal{L}(x)$ 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 the corresponding set of property names. The tree is composed starting from the root node by applying specific expansion rules which are extending the label of existing nodes, adding new nodes or merging existing nodes in the graph. For instance, the application of the $\rightarrow \exists$ rule (see [1]) to the expression $\exists R.C$ in a node x leads to a new node y with a new edge xRy and the extension of the label of y to $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$. In case of non-determinism (e. g. disjunction or node merging) the algorithm will process the possible alternative expansions. The algorithm stops processing a node if no more rules are applicable or if a contradiction (so-called clash) has been found in the node.

Definition (clash) A completion graph G is said to contain a *clash* for some node x in G with respect to a Tbox T if:

- for some A, $\{A, \neg A\} \subseteq \mathcal{L}(x)$;
- $\{(\geq\ m\ S), (\leq\ n\ R)\} \subseteq \mathcal{L}(x)$ with an arbitrary sub-property S of R ($S \sqsubseteq R$) in T; $n, m \in \mathbb{N}; m > n$;
- $\{(\leq\ n\ S)\} \subseteq \mathcal{L}(x)$ and if x has $\geq n+1$ distinct S-neighbours, where a node y is called a S-neighbour of x if y is a S-successor of x or x is a S^{-}-successor of y;
- for some $o \in N_I$, there are $x \neq y$ with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$.

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. DL-style tableau algorithms for $\mathcal{SHOIN}$ are proven to be sound and complete [11].

3 Tableau Inference Illustration

The tableau calculus is based on the refutation principle, whose objective is to prove a formula by showing that its negation cannot be satisfied. For example, the query $A \sqsubseteq C$ will be proven by deriving the unsatisfiability of its complement, namely $A \sqcap \neg C$. We assume that humans typically do not conclude this way. In addition the tableau algorithms used in well-known reasoning systems such as Pellet [20] or RacerPro [10] make use of elaborated optimization procedures which require syntactical transformations obscuring the structure of the original query. For instance, the standard tableaux algorithm does not need to distinguish between the following two subsumption queries: $A \sqsubseteq B \sqcup C$ and $A \sqcap \neg B \sqsubseteq C$. From a logical point of view both expressions reduce to the negation normal form $A \sqcap \neg B \sqcap \neg C$. For the purpose of explaining they need to be treated as different queries in order to be able to generate customized explanations.

3.1 Tagging

We address this problem by omitting structure-destroying transformations and by applying a so-called tagging [3] technique which labels the subsumer of a subsumption query with a special flag († 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.

For a subsumption query $A \sqsubseteq C$ the right-hand side is tagged in the corresponding refutation problem, i. e. $A \sqcap \neg C^{\dagger}$. To generate a user-consumable explanation, all tagged expressions have to be negated again to receive the original query. For instance, the tagged expression $A \sqcap C \sqcap \neg A^{\dagger}$ corresponds to the original query $A \sqcap C \sqsubseteq A$. The latter can be explained by the statement: "*A and something is subsumed by A*".

3.2 Drill-down Explanations

Our approach assumes that humans usually drill down into a problem by reducing it into smaller and more tractable pieces by trying to keep the overall structure of the original problem at the same time. To meet this style of deduction we explain a subsumption relationship by breaking it down into sub-subsumption queries until they reach a level of triviality. This is in compliance with the well known tableau-technique of *lazy unfolding*, which delays the unfolding of a complex concept definitions until it is required within the proof procedure. When unfolding the definition of a concept its name is replaced by the concept definition.

This process of de-referencing has to be explained to the user, of course. All necessary unfolding steps within one tableau-node are collected in order to explain them in one single step. For example, consider the subsumption query

$Father \sqsubseteq \exists hasChild.Human$

and the following concept definition:

$Father \equiv Man \sqcap \exists hasChild.Human$

Then the definition of $Father$ can be expanded to

$Man \sqcap \exists hasChild.Human \sqsubseteq \exists hasChild.Human$

which leads to the explanation statement:

> We have to check whether
> $Father \sqsubseteq \exists hasChild.Human$ holds.
> This is equivalent to
> $Man \sqcap \exists hasChild.Human \sqsubseteq \exists hasChild.Human$
> - by unfolding $Father$ to $Man \sqcap \exists hasChild.Human$

3.3 Explanation tableaux

We use a modified tableau algorithm specifically designed to support explaining. Therefore we do not use sophisticated optimization techniques (other than lazy unfolding and simple normalization) because they typically "destroy" the structure of the original query by applying elaborated syntactical transformations.

Furthermore, our algorithm does not abort processing a branch when the first clash is found. This has the background that a subsumption query can possibly be explained in multiple ways which are represented in the tableau proof by alternative closed branches. For this reason we process the tableau exhaustively by collecting all possible contradictions. Each of them corresponds to an alternative explanation of one and the same inference, which we call an *explanation tableau*. A qualitative evaluation function is then used to rate these options with respect to their adequacy for generating an explanation. In our implementation we use a simple metric which evaluates an explanation tableau ET with the metric $c(ET) =\mid nodes(ET) \mid * bf$ where bf is the branching factor of non-deterministic expansions within an explanation tableau. More sophisticated evaluation metrics are conceivable, of course. In the ideal case alternative explanation tableaux are rated by considering the sum of expected hypothetical intellectual costs of each of its explanation steps.

In case of multiple clashes within one node we rank different types of clashes with respect to their comprehensibleness. For instance, a simple atomic clash such as $\{A, \neg A\} \subseteq \mathcal{L}(x)$ is assumed to be easier to explain than a cardinality conflict $\{(\geq m\ R), (\leq n\ R)\} \subseteq \mathcal{L}(x)$, with $m > n$. In this ordering we also consider which sides (subsumer and/or subsumee) are involved in the clash. For example, $\{C, \neg C\} \subseteq \mathcal{L}(x)$ results in a different explanation than $\{C, \neg C^{\dagger}\} \subseteq \mathcal{L}(x)$. For a detailed description of clash types see [18]. This metric is embedded into the tableau algorithm. Therefore, when testing for a contradiction we search according to the order of clash types so that we can abort processing a node after the first clash.

While building up the tableau proof tree we make use of annotated terms to label concepts whenever they contribute to a clash. We also keep book of dependencies between nodes and expressions to save the information by which rule a node/expression has been introduced. Once a tableau has been selected for explaining we extract the relevant nodes and expressions out of the tableaux and annotated terms. This allows to hide irrelevant expressions in the explanation of an entailment as well as to reduce the length of an arbitrary explanation. For this we introduce for an explanation tableau the mutually dependent notion of relevance of nodes and of expressions.

Definition (relevant node, expression) A node n of an explanation tableau ET is *relevant* iff it contains a relevant expression. An expression t of a node x of ET is *relevant* iff

- t is part of a clash, e.g. $t \in \{A, \neg A\} \subseteq \mathcal{L}(x)$;
- $t \in \{\exists R.C, (\leq 1\ R)\}$ and x has a successor node y arising from the application of one of the rules $\rightarrow \exists$, $\rightarrow \leq$ to t and y is relevant;
- a subexpression t' of t is introduced at an arbitrary successor node y of x by the rule $\rightarrow \forall$ or $\rightarrow \forall_+$ and t' is relevant in y.
- unfolding of an expression t introduces a subexpression t' which is relevant in x.
- $t = C \sqcup D$ and the explanation tableau has two successor nodes generated by applying rule $\rightarrow \sqcup$ to t and both C and D are relevant in the respective successor node.

Figure 1 illustrates a tableau containing two explanation tableaux for the following subsumption of

$$\exists R.C \sqcap \exists R.D \ \sqsubseteq\ (\exists R.A) \sqcup \exists R.C$$

given the TBox-axiom $D \sqsubseteq A$, where $\times$ flags clashing concepts.

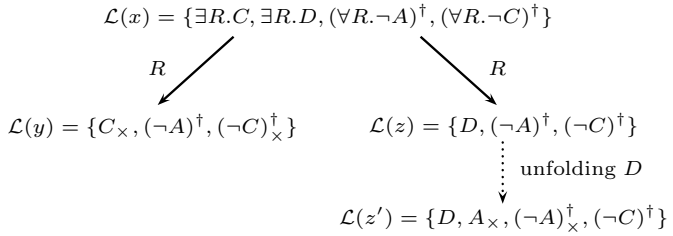

Figure 1: Tableau with two possible explanations

The tableau in figure 1 contains the explanation tableaux $< x, y >$ and $< x, z >$. When selecting the first option for explaining we identify the relevant expressions within $< x, y >$, which are marked by $\times_r$ in figure 2. The subsumption can

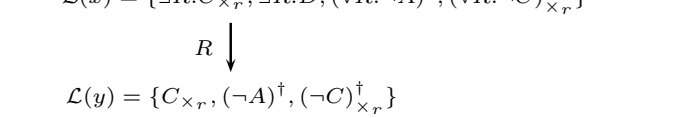

Figure 2: Identifying relevant expressions

be explained by the following statement, where all irrelevant expressions are hidden:

> We have to check whether
> $\exists R.C \sqcap \exists R.D \sqsubseteq (\exists R.A) \sqcup \exists R.C$ holds.
> For the property R we have to check whether
> $C \sqsubseteq \ldots \sqcap C$ holds.
> The subsumption holds since C is subsumed by C and something.

4 Explaining and Patching

4.1 Explaining tableau rule applications

The explanation is generated by traversing an explanation tableau, whereas each application of a tableau rule results in one or more explanation steps which are represented using canned text patterns. We distinguish between explanations of nodes and explanations of edges of an explanation tableau. In the following we will shortly introduce a selection of explanation steps, for a detailed description we refer to [18, 14].

Existential Requirements. First we examine the explanation of edges in an explanation tableau, which are introduced by an existential restriction such as $\exists R.A$ or $(\geq 1\ R)$. Consider the existential quantification $\exists R.C$ over a property R, which results in a R-successor node containing the expression C. The R-successor node will be introduced by an explanation step with help of a text fragment like *"For the property R we have to check whether $C \ldots \sqsubseteq \ldots$ holds"*. The new node then may contain additional expressions other than C due to other restrictions (e. g. $\forall R.D$). If a tagged expression is involved we need to generate an explanation with respect to the original inference problem, e. g. for some node x with $\mathcal{L}(x) = \{\exists R.C, A, \forall R.\neg D^{\dagger}, A^{\dagger}\}$ we regain the original inference problem $\exists R.C, A, \exists R.D \sqcup A$ by negating all tagged concepts.

To give an example consider the following subsumption query: $\forall hasChild.Male \sqcap (\geq 1\ hasChild) \sqsubseteq \exists hasChild.Male$. The approach we described up to now is able to produce the following explanation:

> We have to check whether $\forall hasChild.Male \sqcap (\geq 1\, hasChild) \sqsubseteq \exists hasCild.Male$ holds.
> The subsumption holds because we can show for the property $hasChild$ that $\ldots \sqcap Male \sqsubseteq Male$ holds.
> This because it holds that $Male$ and something is subsumed by $Male$.

Cardinality Restrictions and Merging. A feature of $\mathcal{SHOIN}$ is the ability to express cardinality restrictions, i.e. to constrain the number of possible fillers for an arbitrary property. We will shortly introduce the explanation of two contradictory cardinality restrictions. Hereby we have to take the origin of the expressions into account (lhs vs. rhs) and therefore have to distinguish four types of cardinality clashes. Each of them results in a different explanation statement:

- $(\leq n\ R) \sqcap (\geq m\ R) \sqsubseteq \ldots$, where $m > n$
  "The expression $(\leq m\, R) \sqcap (\geq n\, R)$ is equivalent to $\bot$, since there can't be at-least m and at-most n fillers for the property R. The concept $\bot$ is subsumed by everything."
- $\ldots \sqsubseteq (\leq n\ R) \sqcup (\geq m\ R)$, where $m \geq n + 1$
  "The expression $(\leq\ m\ R) \sqcup (\geq\ n\ R)$ is equivalent to $\top$, since there are allways less or equal than m, or more or equal than n fillers for the property R. The concept $\top$ subsumes everything."
- $(\leq n\ R) \sqsubseteq (\leq m\ R)$, where $m \geq n$
  "At most n fillers for the property R is subsumed by at most m fillers for the property R."
- $(\geq m\ R) \sqsubseteq (\geq n\ R)$, where $m \geq n$
  "At least m fillers for the property R is subsumed by at least n fillers for the property n."

Furthermore, merging results from a combination of m existential quantifications over a property R and a $(\leq n\ R)$ restriction with $m > n$. This forces a property R to have at-most n fillers over R and requires the non-deterministic merging of all existing successor nodes to at-most n nodes. As an example consider the expression $\exists R.A \sqcap \exists R.B \sqcap (\leq 1\ R) \sqsubseteq \exists R.(A \sqcap B)$. The result of the at-most restriction leads to the explanation: "Since there has to be one filler for for each of the classes A and B and the property R is restricted to at-most 1 filler on the lhs, we have to check for the property R whether $A \sqcap B \sqsubseteq A \sqcap B$ holds."

Property Hierarchies and Property Restrictions. Properties can be arranged in a so-called property hierarchy. Each property filler within a property hierarchy is also a filler of all its super properties. E. g. if the property son is a sub-property of $child$, then each filler of son is also a filler of $child$. Consider a sub-property S of R ($S \sqsubseteq R$) and an expression $\forall R.A \sqcap (\geq 1\ S)$. Each relevant S-successor node will be introduced by the additional explanation statement that *"All fillers of R are restricted to be of the class A. Since S is a sub-property of R this restriction also applies to S."* In case of a clash between cardinality restrictions involving a sub-property the explanation is analogous to the explanation of cardinality restrictions above. As example consider the expression $(\leq n\ R) \sqsubseteq (\leq m\ S)$ with $m \geq n$ and $S \sqsubseteq R$, which leads to the following explanation: *"At most n fillers for the property R is subsumed by at most m fillers for the property S, which is a sub-property of R."*

OWL allows to restrict properties in their global domain and range or to declare a property to be functional or transitive. In addition, properties can be defined as sub-properties of other properties. Due to the lack of space we will only consider an example of domain/range restrictions here. Consider a given domain or range restriction of a property which will be added to the successor resp. predecessor node by the tableau algorithm. The restriction can consist of any valid $\mathcal{SHOIN}$ expression. In case of relevance of the introduced expression in the successor resp. predecessor node, the range resp. domain restriction has to be explained by an additional statement. As example consider the expression $(\geq 1\ R) \sqsubseteq \exists R.B$ with a range restriction C on R. The explanation for the edge to the successor node is then as follows: *"For the property R exists the range restriction C that has to be considered on the left hand side. We have to show for the property R whether $C \sqsubseteq B$ holds."*

Optimizations It is important to generate explanations as simple to understand as possible. Therefore we have implemented several optimizations which condense the explanation in specific situations called *filtering*, *mode-switching* and *aggregating*.

Filtering is a simple method to prevent non-deterministic expansions within the tableau with the help of structural comparison. E.g. the standard tableau algorithm would split the disjunction within the subsumption $A \sqcap B \sqcap C \sqsubseteq A \sqcap C$ which is represented in the tableau as $\{A, B, C, \neg A^{\dagger} \sqcup \neg B^{\dagger}\}$. This is not necessary and the subsumption trivially holds, since the subsumer is a specialization of the subsumee. Therefore our algorithm does a structural comparison for each node in the

tableau. An obvious subsumption is found if the subsumer is a syntactical subset of the subsumee.

Mode-switching is used in situations where the subsumer or subsumee of a tableau node is unsatisfiable by its own. In this case either the subsumer is equivalent to $\top$ or the subsumee is equivalent to $\bot$. In this case our explainer will switch to unsatisfiability respectively tautology explaining while omitting the irrelevant side.

Aggregating is a method to summarize similar consecutive explanation steps into one summary statement. The user is free, however, to expand the aggregated step into its single pieces on demand. As an example consider the expression $\exists R.\exists R.\exists R.C \sqsubseteq \exists R.C$ with property R as transitive relation. This would lead to a chain of similar explanation statements for the R-successors. After aggregating the explanation for the successor nodes is as follows: *"For a sequence of successors over the property R we have to check whether $C \sqsubseteq C$ holds."* Optionally the user can expand the aggregating explanation step to see all details , for the example this yields to:

For a sequence of successors over the property R we have to check whether $C \sqsubseteq C$ holds. Because property R is transitive we have to consider C on the rhs.

- Check for the property R if $\exists R.\exists R.C \sqsubseteq \exists R.C$ holds.
- Check for the property R if $\exists R.C \sqsubseteq \exists R.C$ holds.
- Check for the property R if $C \sqsubseteq C$ holds.

4.2 Explaining & Patching Non-Subsumption

When applying the techniques from above a non-subsumption between A and C (written $A \not\sqsubseteq C$) results in at least one model for the corresponding negated subsumption query (i.e. $A \sqcap \neg C$). Technically, each of these models is caused by an unclosed (non-clashing) subgraph in the tableau and can be interpreted as a counter example with respect to the subsumption query.

The idea of explaining non-subsumption is quite similar to the approach of explaining subsumption. All explanations of unclosed paths of the tableau build up the explanation for the non-subsumption. Hereby an explanation ends when there are no more successors which were generated and constrained by the rhs and lhs of a node. An exception is trivial non-subsumption ($lhs \equiv \bot$ or $rhs \equiv \top$).

To patch a non-subsumption we have to fill a logical gap to establish a desired subsumption. The problem with patching a non-subsumption is the infinite search space caused by the infinite many ways of closing at least one of the potentially many unclosed tableau subgraphs. Consider the (worst) case of two arbitrary and unrelated concepts. Obviously there are uncountably many options in order to establish a subsumption relationship between them. Even if not all are sensible at all or trivial (e. g. adding the subsumer to the subsumee) patching non-subsumption apparently is difficult. Therefore, our method is to constrain the search space by concentrating on a set of common errors of inexperienced users which were studied in [17]. The errors from [17] considered in this approach are: all-quantification instead of existential quantification ($\forall$ for $\exists$), wrong use of negation in combination with a quantor ($\neg\forall r.(\ldots)$ vs. $\forall r.(\neg\ldots)$ and $\neg\exists r.(\ldots)$ vs. $\exists r.(\neg\ldots)$) and use of a partial instead of a complete definition. In addition we consider the possibility of a missing term in a definition.

Our algorithm searches for symptoms of the described errors in every node of an unclosed path of an explanation tableau. For instance, nodes with an expression containing a common (sub-)relation from both sides might be introduced by one of the quantor-related errors, e. g. in $\exists son.Father \not\sqsubseteq \forall child.Father$, the all-quantifier matches a common user error as described in [17]. When observing such an expression in a node, we investigate all involved concept definitions in order to identify one of the quantor-related errors.

Although, concentrating on the mentioned errors reduces the search space significantly, this approach still leads to many patch suggestions. To further reduce the search space we filter the remaining suggestions. At first we drop those suggestions which lead to a trivial (partial) subsumption similar to the approach mentioned in [12]. Our approach, furthermore, assumes that the user wants to patch the non-subsumption but does not want the class hierarchy to change elsewhere (compare with [8]). As a metric we identify the graph edit costs by computing the edit distance between the ontology hierarchy before and after applying the suggested patch. The cost of a suggestion are computed by counting the amount of modifications a suggestion would cause within the direct sub- and super-concepts for all concepts of the ontology. For every possible suggestion the effect of the modification is calculated by finding out whether it would patch the non-subsumption and by computing the metric. A suggestion patching the non-subsumption is rated better than one which does not patch it (but might be one right step for patching it). Secondly, a suggestion with little impact on the hierarchy is rated better than one with a high impact.

5 Implementation

Explaining an inference can significantly improve the authoring process of an ontology. Consequently, explanations are most powerful when combined with an ontology editing tool. Therefore, both non-standard inference services, namely explaining as well as patching, have been implemented as described above and added as plug-ins to our ontology authoring tool OntoTrack [15].

Explanation of (non-)subsumption in OntoTrack is available on user demand. The user can select two arbitrary concepts within the graph hierarchy layout via context menue. The explanation component then offers a tree-like expand list which allows to drill down the explanation path. In case of a non-subsumption all red colored explanation steps show those parts of the explanation which end up in a non-subsumption. Those parts also offer a patch suggestion via a "**?**" button. The patch suggestions are sorted according to their rating and those which completely patch the non-subsumptions are colored green.

Figure 3 shows the explanation of *"$VegetarianPizza$ is not subsumed by $MeatyPizza$"* within the well-known pizza ontology [7]. The explainer has to show for each of the individual parts on the rhs, that they are included in the lhs, which is shown in the cases $1-5$. The cases $1-4$ explain valid partial subsumption whereas the proof fails for case 5 and therefore the subsumption does not hold. To establish the subsumtion the user can ask for patch suggestions (by clicking the Button "?"), which are shown in Figure 4 for our example.

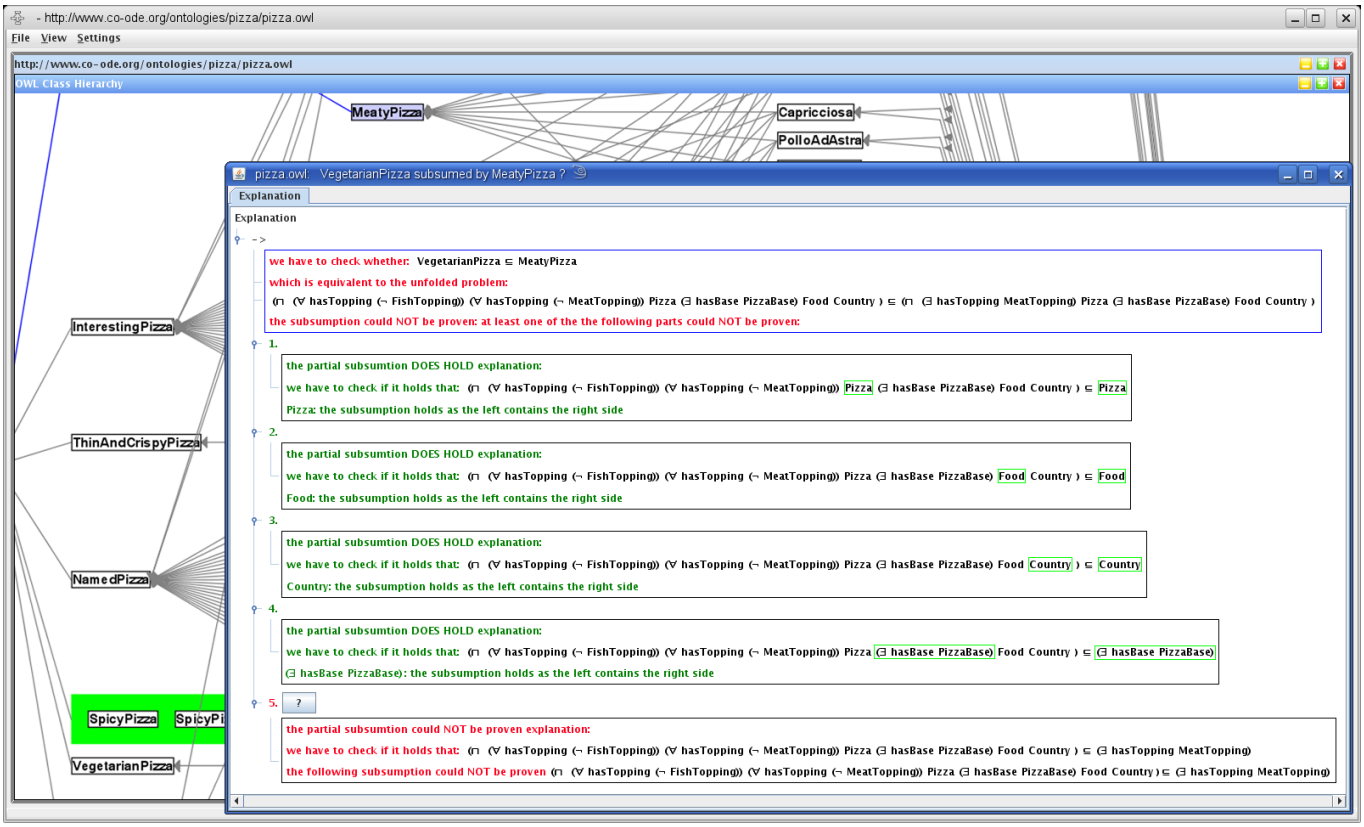

Figure 3: Example of explaining and patching in Ontotrack

6 Related Work

The very first approach for explaining DL inference services such as subsumption relied on structural algorithms providing proof fragments within DL systems like Classic [16] or (Power)Loom [4]. A different approach [3] suggests to utilize a sequent calculus to explain the trace of a tableaux proof for the DL $\mathcal{ALC}$. An implementation of the latter is given in [13] which also introduced the so called dual-reasoner architecture which helps to identify non-relevant parts of the tableaux tree by querying an optimized external reasoner. A different approach tries to explain a subsumption by presenting an intermediate concept between subsumer and subsumee [19]. Closely related to the latter is [12] which aims at explaining unsatisfiability by computing the Minimal Unsatisfiability Preserving Sub-TBoxes (MUPS) to find potential debugging clues. Another approach is based on resolution algorithms [6] where an explanation for an ABox entailment is compiled from the deduction trace of a resolution prover.

7 Discussion and Outlook

We argue that tableau-based methods are valuable for explaining well as providing clues for repairing an unwanted non-subsumption. Concerning explanations we have extended previous work with GCI's and optimization techniques such as hiding irrelevant expressions and aggregating similar explanation steps. As a proof of concept we implemented an explanation component suitable to deal with the language $\mathcal{SHF}$ which has been integrated into the ontology authoring tool ONTOTRACK. In comparison to axiom pinpointing our method is currently restricted in language expressivity but has the benefit of being applicable even to large ontologies without necessarily running into computational complexity problems. An extension to explaining Abox-entailments also seems possible.

The task of generating suggestions for patching an unwanted non-subsumption obviously is more difficult. Finding and selecting solutions for obtaining a subsumption relationship has many, partially conflicting dimensions such as minimality or conservativeness [8]. Our approach to identify common user errors collected by [17] allows to significantly reduce the search space but is limited to certain types of errors and a combination of several typical flaws might not be found. Rating different suggestions is difficult and requires the development of more fine-grained metrics. An interesting idea for future work is to distinguish between different types of axioms (e. g. to assume the definitions of some base ontologies to be bug-free) [8]. A further idea is to rate patches with respect to the change history of the definition they refer to (a kind of "latest-first" conflict resolution strategy for suggestions). Nevertheless, any debugging or patching service can only considered as some proposal. These non-standard inference services, therefore, should be considered as helpful hints in the course of developing and maintaining high-quality ontologies.

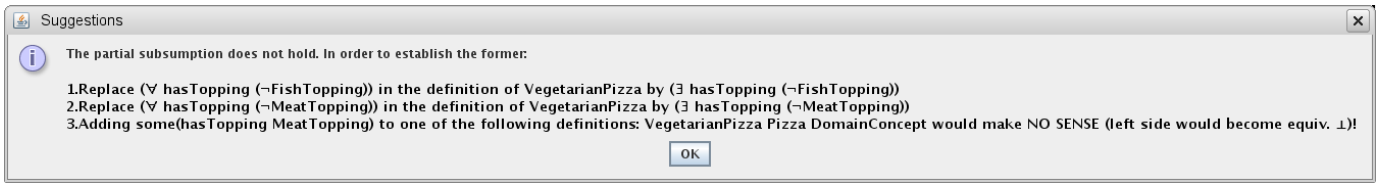

Figure 4: Patch suggestion for "$VegetarianPizza$ is not subsumed by $MeatyPizza$".

References

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language reference, W3C recommendation. Technical report, World Wide Web Consortium, Februar 2004.

[3] A. Borgida, E. Franconi, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. Explaining $\mathcal{ALC}$ subsumption. In *Proc. of the Int. Workshop on Description Logics (DL99)*, pages 37–40, Linköping, Sweden, 1999.

[4] H. Chalupsky and T. Russ. WhyNot: Debugging Failed Queries in Large Knowledge Bases. In *Proc. of the Innovative Applications of Artificial Intelligence Conf. (IAAI-02)*, pages 870–877, Edmonton, AL, Canada, 2002.

[5] Mills Davis. Semantic Wave 2006: Part-1. Technical report, Project10X, Washington, DC, USA, 2006.

[6] X. Deng, V. Haarslev, and N. Shiri. Resolution Based Explanations for Reasoning in the Description Logic $\mathcal{ALC}$. In *Canadian SW Working Symposium (CSWWS06)*, volume 2, pages 189–204, 2006.

[7] N. Drummond, M. Horridge, R. Stevens, C. Wroe, and S. Sampaio. Pizza Ontology, http://www.co-ode.org/ontologies/pizza, 2007.

[8] C. Elsenbroich, O. Kutz, and U. Sattler. A Case for Abductive Reasoning over Ontologies. In *Proc. of OWL: Experiences and Directions WS*, 2006.

[9] Noah S. Fiedland, Paul G. Allen, Michael Witbrock, Gavin Matthews, Nancy Salay, Pierluigi Miraglia, Jürgen Angele, Steffen Staab, David Israel, Vinay Chaudhri, Bruce Porter, Ken Barker, and Peter Clark. Towards a Quantitative, Plattform-Independent Analysis of Knowledge Systems. In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning*, pages 507–514, Whistler, BC, Canada, June 2004. AAAI Press.

[10] V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, pages 27–36, 2003.

[11] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):S. 239–264, 2000.

[12] A. Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, University of Maryland College Park, 2006.

[13] F. Kwong. Practical approach to explaining $\mathcal{ALC}$ subsumtion. Master's thesis, University of Manchester, 2005.

[14] Julian Lambertz. Erklärung und Korrektur von Nicht-Subsumtion in Ontologien. Master's thesis, University of Ulm, 2007.

[15] Thorsten Liebig and Olaf Noppens. OntoTrack: A Semantic Approach for Ontology Authoring. *Journal of Web Semantics*, 3(2):116–131, 2005.

[16] D. McGuinness and A. Borgida. Explaining Subsumption in Description Logics. In Chris Mellish, editor, *14th International Joint Conference on Artificial Intelligence*, pages 816–821, 1995.

[17] A. L. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In *EKAW*, pages 63–81, 2004.

[18] Stephan Scheele. Tableaubasierte Ableitung und Erklärung von Subsumtionsbeziehungen. Master's thesis, University of Ulm, 2007.

[19] S. Schlobach and R. Cornet. Explanation of terminological reasoning: A preliminary report. In *Description Logics*, 2003.

[20] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A Practical OWL-DL Reasoner. *Journal of Web Semantics*, 2006.

Kontakt

Dr. Thorsten Liebig
Institute of Artificial Intelligence
Ulm University, D-89069 Ulm
Email: Thorsten.Liebig(AT)uni-ulm.de

Stephan Scheele
University of Bamberg
Feldkirchenstr. 21, D-96045 Bamberg
Email: stephan.scheele(AT)uni-bamberg.de

Bild

Thorsten Liebig is a scientific assistant at Ulm University. In 2000 he received his Dr.-Ing. from Otto-von-Guericke University, Magdeburg. He has received scholarships from the state of Sachsen-Anhalt as well as the German Academic Exchange Service (DAAD). In 1997 he was a visiting scientist at the University of Southern California in Load Angeles. His research interests are in Description Logics, ontology authoring, as well as reasoning techniques.

Bild

Stephan Scheele studied computer science at the University of Ulm. Since 2007 he is a PhD student of the Informatics Theory Group at the University of Bamberg. His current research is devoted to semantic technologies and formal methods in the field of auditing.