Explanation Support for OWL Authoring

Thorsten Liebig and **Friedrich von Henke** and **Olaf Noppens**
Dept. of Artificial Intelligence, Faculty of Computer Science
University of Ulm, D-89069 Ulm, Germany

Abstract

Recent analyses of ontology engineering efforts showed that it is not only novices which are commonly faced with comprehension problems or misconceptions of modeling consequences. A better understanding can be fundamentally supported by on-demand explanations of subsumption, the core inference service of ontology reasoning systems. In this paper we describe ongoing work towards a systems for explaining subsumption for a significant fraction of the Description Logic (DL) underlying OWL Lite. Our explainer uses a tableau-based approach to generate step by step quasi-natural language explanations. We also comment on related work and discuss future explaining features.

Introduction

The Web Ontology Language (OWL) recently has been approved as a W3C Recommendation (Bechhofer *et al.* 2004) for the publication and the exchange of ontological models on the Web. Two of the three sublanguages of the specification, namely OWL Lite and OWL DL, are based on Description Logics (DLs). This allows to exploit many results of a long time of KR research, such as formal properties like decidability and complexity of key inference problems, sound and complete algorithms, as well as implemented reasoning systems (Horrocks 2005).

Without doubt, practically efficient reasoning algorithms for exploiting the semantics captured in ontologies are an important prerequisite for OWL based applications. The most important sub-languages of OWL, namely Lite and DL, are quite expressive. Therefore a practical decision procedure that is likely to perform well with realistic ontologies has only just been published (Horrocks & Sattler 2005). To have standard reasoning services is, however, only one part of the story. Knowledge intensive applications critically depend on the existence of sound and well-balanced ontologies. Unfortunately, less effort has been spend to develop appropriate mechanisms in order to support ontology engineers to design and maintain such ontologies.

This seems to be a major shortfall, because experience has shown that authoring ontologies is a difficult task not only for people without much formal background. As an example, in the course of teaching OWL the developers of Protégé report on many typical comprehension problems for novices (Rector *et al.* 2004): users often use an universal instead of an existential quantification because of a misconception of the semantics, have the incorrect assumption of classes being disjoint by default, are not aware about the effect of domain or range restrictions as well as transitive properties on reasoning, etc. Surprisingly its not only non-sophisticated user which are faced with ontology modeling problems of this kind. A failure analysis of three different efforts of formalizing knowledge for a given domain (Fiedland *et al.* 2004) showed that even Knowledge Representation (KR) experts often have a misunderstanding of the inference algorithm or fail to model implicit context assumptions.

Providing Instant Reasoning Feedback

In order to better assist users with the task of ontology authoring at least two supporting services are required. First, authoring tools should interweave modeling and reasoning more tightly by providing instant reasoning feedback during editing. Making implicit modeling consequences explicitly and instantly available helps users to check for imprecise, redundant or faulty modeling. Unfortunately this kind of direct feedback is rarely found in current ontology editors. Notable exceptions are the Swoop editor (Kalyanpur, Parsia, & Hendler 2005) and our ontology tool ONTOTRACK (Liebig & Noppens 2005).

Explaining Reasoning Results

Second, as a natural consequence of the first, tools should be able to explain those reasoning results in a fashion which allows users to follow the conclusion. It is commonly known that most consequences are not easily traceable or intuitive. Some users may even mistrust the inference results because of unintended outcomes. This is due to the fact that reasoning consequences often depend on logical interrelations with other definitions or nested definitions, which themselves may depend on other consequences.

Obviously, the ability to explain how and why a result has been derived would significantly improve the ontology development process. Such a service would support a user in understanding an ontology by decoding the semantical consequences of the ontology axioms. This in turn enables a user to make more deliberate modeling decisions because any unintended outcome would become immediately explicit and comprehensible. A user could then easily step back, revise recent modeling choices and proceed.

Outline

In our vision of upcoming tools for the Semantic Web we see ontology editing tools shifting from a syntactical centered to a semantically centered environment. Obviously, such an ontology authoring tool not only has to provide a convenient browsing interface or intuitive editing capabilities, but also has to support the user in understanding the logical meaning of an ontology. Consequently, the explicit representation of the semantical consequences as well as how they are derived should become an integral part of any serious ontology authoring tool.

As a conclusion from the above, ontology authoring presumably is not possible without a basic understanding of the underlying reasoning services. This paper provides an overview about the conceptual approach as well as implementation issues of our explaining system which aims at supporting a deeper comprehension of subsumption, the core inference service for ontologies. This is followed by an discussion of related work and an outlook with some thoughts about potential interesting explaining features for ontology authoring.

OWL – The Web Ontology Language

OWL is layered on top of other fundamental Web language standards. The lowermost layer in terms of language standards is XML. XML can be seen as the transport layer of the higher-ranking languages. On top of XML there is an abstract data model called Resource Description Framework (Klyne & Caroll 2004), RDF for short. Above RDF is RDF Schema (RDFS) (Horrocks, Patel-Schneider, & van Harmelen 2003) adopts the basic fact-stating ability of RDF and provides lightweight class- and property-structuring capabilities. At last, OWL is layered on top of RDFS. OWL comes with three increasingly expressive sub-languages: OWL Lite, OWL DL, and OWL Full. Within OWL one can define classes and properties, and organize them in a hierarchy, as can RDFS. In contrast to RDFS, OWL classes can be specified as logical combinations of other classes, or as an enumeration of individuals. As a notational convention we use upper case symbols A and B for atomic classes and C, D, and E to denote classes with complex definitions in the following. Properties will be denoted by lower case letters like r, p, q, etc. Beyond that we use the abstract DL-style notation (Baader *et al.* 2003) for our examples in the following.

The most expressive but still decidable sub-language is OWL DL. Within OWL DL a complex class description can be an enumeration of individuals $\{i_1, \ldots, i_m\}$ or composed of other descriptions or restrictions by using operators like union ($\sqcup$), intersection ($\sqcap$), and complement ($\neg$). A restriction can consist of a qualified universal quantification over a property ($\forall r.C$), a qualified existential quantification ($\exists r.C$), or a cardinality expression ($\leq \mid \geq \mid = n\ r$) with $n \in \mathbb{N}$. Their semantics is defined by an interpretation

Syntax	Semantics
A	$A^{\mathcal{I}}$
$\top$	$\Delta^{\mathcal{I}}$
$\bot$	$\emptyset$
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$\{i_1, ...\}$	$\{i_1^{\mathcal{I}}, ...\}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}}, \text{if } \langle x, y\rangle \in r^{\mathcal{I}}, \text{then } y \in C^{\mathcal{I}}\}$
$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} \text{ with } \langle x, y\rangle \in r^{\mathcal{I}} \text{and } y \in C^{\mathcal{I}}\}$
$(\leq n\ r)$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid \langle x, y\rangle \in r^{\mathcal{I}}\} \leq n\}$
$(\geq n\ r)$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid \langle x, y\rangle \in r^{\mathcal{I}}\} \geq n\}$
$(= n\ r)$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid \langle x, y\rangle \in r^{\mathcal{I}}\} = n\}$

Table 1: OWL DL abstract syntax and semantics

$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with an interpretation function $\cdot^{\mathcal{I}}$ and an interpretation domain $\Delta^{\mathcal{I}}$. The interpretation function maps every concept to a subset of $\Delta^{\mathcal{I}}$ as shown in table 1 ($\sharp$ denotes set cardinality). An individual i_k is an object of the interpretation domain ($i_k^{\mathcal{I}} \in \Delta^{\mathcal{I}}$). Since OWL Lite is a syntactical subset of OWL DL it does not allow for unions, complement, enumerations, and cardinalities greater than 1 (grayed columns in table 1). However, the syntactical restrictions of OWL Lite come with relatively little loss of expressive power. In concrete, with the help of indirection and multiple definitions for a single class identifier, all of OWL DL can be captured in OWL Lite except those descriptions containing either individuals or cardinalities greater than 1 (Horrocks, Patel-Schneider, & van Harmelen 2003).

In OWL DL as well as Lite a property can be declared as transitive, symmetric, functional, or as inverse of another property. Semantically properties are mapped to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The domain and range of a property can be globally restricted within both sub-languages.

Explaining Subsumption within OWL Lite

Our work currently is focused on providing on-demand quasi-natural language explanations of subsumption, the core inference service of ontology reasoners. A class C is said to subsume a class A, written $A \sqsubseteq C$, iff the set-theoretic interpretation of the former is a subset of the latter ($A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$), or iff the left hand side (lhs) implies the right hand side (rhs) when using a translation into predicate logic. With respect to class centered formalisms the subsumption relationship corresponds to the sub-class relation, i. e. A is explicitly or implicitly a sub-class of C. Subsumption is the central relationship within ontology reasoning. The most other interesting inference services do naturally depend on the subsumption relationship. For example, equality between classes is defined as mutual subsumption ($A \doteq B$ iff $A \sqsubseteq B$ and $B \sqsubseteq A$).

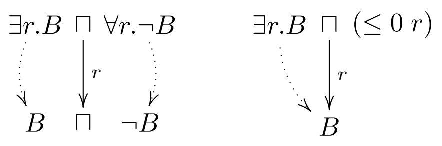
The main idea of our approach is to compile explanation steps out of text patterns which are triggered by a proof algorithm typically used for ontology reasoning. The system we are currently working on implements and extends the idea of adapting a DL tableau algorithm for explaining as suggested in (Borgida *et al.* 1999). We decided to build our own system called MEX (Liebig & Halfmann 2005), since currently available tableaux reasoners are either not available as source code or are based on highly optimized algorithms not easily extensible for explanation generation.

Our goal is to build an explainer capable of explaining subsumption within the most reasonable subset of OWL Lite, which we call OWL Lite$^-$ (Liebig & Noppens 2004). Currently the system is able to generate explanations for the subsumption relationship within definitorial $\mathcal{ALEHF}_{R^+}$ ontologies, i. e. ontologies with unique acyclic definitions using conjunction, qualified existential and universal quantification, atomic negation, property hierarchies, and a limited form of cardinality restrictions. Although not explicitly present in our language, disjunction comes implicitly due to the proof strategy of the tableau algorithm. In order to cope with all of OWL Lite$^-$ we solely have to add inverse properties (see Discussion and Outlook).

DL-style Tableaux Algorithms

As mentioned before, our approach relies on a tableau proof algorithm. Due to the lack of space we will only sketch the general procedure of DL-style tableaux algorithms and refer to (Baader *et al.* 2003) for a detailed description. Tableaux algorithms systematically try to construct a model of an expression. The initial expression builds the root node of a so-called tableau tree and may correspond to queries such as class satisfiability or subsumption. Specific expansion rules successively decompose the nodes of such a tree by adding new nodes, append expressions to nodes, or merge nodes. For example, an existential quantification $\exists r.C$ over a property r will result in a so-called r-successor node containing the expression C. In case of non-determinism due to disjunction or merging the algorithm will process all possible alternative expansions. The algorithm stops to process a node when either no more expansion rules are applicable or a clash is discovered. A clash is a contradiction caused by the occurrence of some expression C and $\neg C$ within a node. A clash may also be caused due to a conflicting quantity of successors given in cardinality expressions (e. g. $(\leq 0\ r) \sqcap (\geq 1\ r)$). When all possible expansions will lead to an clash, the root node is proven to be unsatisfiable. It has been shown that tableaux algorithms are sound and refutation complete for the DLs underlying OWL Lite and OWL DL. This means that the tableaux calculus is sound and complete for any kind of unsatisfiability proofs.

The following example shall illustrate the operation method of a DL tableau algorithm. Consider proving $\exists r B \sqcap (\forall r.\neg B \sqcup (\leq 0\ r))$ to be unsatisfiable. Due to the disjunction we need to consider two alternative expansions which are depicted as separate trees in figure 1. The left expansion in figure 1 will lead to a clash because of the presence of B and its complement within the leaf node. The right one will also clash because of conflicting restrictions about the number of r-successor nodes (at least one against at most zero). In consequence of both results the root node is proven to be unsatisfiable since we have shown that all possible expansions will lead to a clash.



$\exists r.B \sqcap \forall r.\neg B$
r
$B \sqcap \neg B$
$\exists r.B \sqcap (\leq 0\ r)$
r
B

Figure 1: A simple DL-style tableau proof.

OWL class (or property) definitions typically consists of expressions which contain other named classes of the ontology. During node processing a tableau algorithm needs to replace those references with their given definition. This process is called unfolding. A commonly used optimization strategy is *lazy unfolding*. Lazy unfolding delays the unfolding of a class to its given definition until it is required by the proof algorithm to proceed. This also helps to maximize performance in typical ontologies since only those axioms are taken into account which are actually proof relevant.

Tableau-based Explaining

The tableau calculus (just as the resolution calculus) employs a refutation strategy which aims to prove a fact or query by showing its opposite to be contradictory. Obviously, this is not an intuitive approach in order to explain *how* a subsumption query has been derived. For example, consider the query $(= 0\ r) \sqsubseteq (\leq 1\ r)$ which will be demonstrated by a refutation prover by showing the unsatisfiability of its complement, namely $(= 0\ r) \sqcap (\geq 2\ r)$ which is the negation normal form of $\neg((= 0\ r) \sqsubseteq (\leq 1\ r))$. To derive the former by showing the latter to be contradictory is typically not used by humans to arrive at a conclusion.

In addition, optimized proof systems often make use of elaborated syntactical transformations which "destroy" the primal structure of the query. Again, this is in apparent contrast to human-style reasoning which usually tries to reduce complex problems into smaller and more comprehensible pieces by keeping the original structure of the problem. Therefore, in order to better meet human-style reasoning, our system explains a subsumption query by breaking it down into sub-subsumption queries until those reveal an obvious subsumption relationship. This procedure is triggered by a tableau proof which decomposes a query by successively applying tableau expansion rules.

Therefore, in comparison to those tableaux algorithms used in commonly known DL systems like Racer (Haarslev & Möller 2001) or Pellet (Parsia & Sirin 2004) our implementation differs at two points. First, we use a technique called tagging which allows to distinguish between the subsumer (rhs) and the subsumee (lhs) of a subsumption query. Second, we do not use sophisticated optimization techniques (other than lazy unfolding) in order to be able to reconstruct the subsumption query out of a tagged expression, for the illustration of the derivation steps at any time of tableau processing.

To given an example consider the following two subsumption queries: $A \sqcap \neg B \sqsubseteq C$ and $A \sqsubseteq (B \sqcup C)$. They both

$$\begin{tikzcd}
\forall \textit{child}.\textit{Male} \sqcap (\geq 1\ \textit{child}) \sqcap [\forall \textit{child}.\neg \textit{Male}]^{\dagger} \arrow[d, "\textit{child}"] \\
\textit{Male} \sqcap [\neg \textit{Male}]^{\dagger}
\end{tikzcd}$$

Figure 2: A simple DL-style tableau proof.

result in the same negation normal form of their corresponding unsatisfiability problem, namely $A \sqcap \neg B \sqcap \neg C$. This means that they collapse into one single query with respect to the tableau procedure. On explanation side, however, they need to be treated as two queries with different explanations. If we would have tagged the first variant ($A \sqcap \neg B \sqcap \neg C^{\dagger}$) the original subsumption statement could easily be reconstructed. Note that in case of unfolding a tagged class its corresponding substitutes also needs to be tagged.

While building up the tableau proof our system generates a corresponding explanation structure in parallel. For each relevant tableau-rule or unfolding application one or more explanation steps are added. An explanation step consists of its corresponding type, its depth in the tableau tree, a list of additional information (relevant classes, nodes, etc.) and a textual explanation. The type and the additional parameters stored within each step can be used by an external component to lay out explanations with respect to the given application context (e. g. degree of detail, hierarchical structure vs. flat ordering, etc.).

As mentioned before, each tableau transformation will result in one or more explanation steps. A tableau transformation consists of unfolding steps as well as tableau rules. As an example, recall our existential quantification $\exists r.C$. The r-successor node will be introduced by our explainer with help of a text fragment like *"For the property r we have to check whether ..."*. The sentence will be finished as soon as the new node has been completely build. Because the new node then may then contain additional expressions (other than C which is implied by the existential quantification) resulting from other restrictions of the current node (see left side of figure 1 for an example). Note that whenever a tagged expression will be involved we need to generate an explanation with respect to the original (i. e. complementary) expression. The same is required to explain a terminating clash accordingly.

To illustrate the above consider the following subsumption query: $\forall \textit{child}.\textit{Male} \sqcap (\geq 1\ \textit{child}) \sqsubseteq \exists \textit{child}.\textit{Male}$. The tableau proof of the corresponding unsatisfiability problem $\forall \textit{child}.\textit{Male} \sqcap (\geq 1\ \textit{child}) \sqcap \neg(\exists \textit{child}.\textit{Male})$ is shown in figure 2. The approach we described so far would be able to produce the following explanation:

> *It holds that* $\forall \textit{child}.\textit{Male} \sqcap (\geq 1\ \textit{child})$ *is subsumed by* $\exists \textit{child}.\textit{Male}$*:*

$\forall \textit{child}.\textit{Male} \sqcap (\geq 1\ \textit{child}) \sqsubseteq \exists \textit{child}.\textit{Male}$

> *Because for the property* child *we can show that* $\top \sqcap \textit{Male}$ *is subsumed by Male:*

$\top \sqcap \textit{Male} \sqsubseteq \textit{Male}$

> *This because it holds that Male and something is subsumed by Male.*

Explanations will be given only to those nodes which are clash relevant. This means that in case of adding non-interfering expressions to the lhs of the previous example, e. g. $\exists \textit{owns}.\textit{Pet} \sqcap \forall \textit{owns}.\neg \textit{Cat}$, the explanation would not grow in size (in the sense of an extra explanation step).

Up to now this section was a description about tableau-based techniques for explaining subsumption within a language with an expressivity of $\mathcal{ALC}$ as conceptually discussed in (Borgida *et al.* 1999). Since we extend previous work in order to deal with $\mathcal{ALEHF}_{R^+}$ we had to add explanations for cardinality restrictions, role hierarchies, merging of role-successors, and domain and range restrictions.

Cardinality Restrictions and Merging. An important language feature of OWL is the ability to express cardinality restrictions (see table 1). Since our goal is to cover OWL Lite we support functional roles and a limited form of cardinality restrictions within our approach. In concrete, we allow for cardinality restrictions $(\geq\ n\ r)$, $(\leq\ n\ r)$, and $(=\ n\ r)$ with $n \in \{0, 1\}$ as well as global constraints restricting a role to have at most one filler exactly as specified within OWL Lite. In fact, the "at least" restrictions are already covered by $\mathcal{ALE}$, because $(\geq\ 0\ r)$ reduces to $\top$ and $(\geq\ 1\ r)$ is equivalent to $\exists r.\top$. In contrast, the "at most" restrictions and the combination of conflicting cardinality restrictions require an extension of the tableau as well as the explanation process. Since we have to take the origin of the restrictions (lhs vs. rhs) into account we have to distinguish between four types of cardinality clashes. Each of which result in a different explanation statement:

- $(\leq n\ r) \sqcap (\geq m\ r) \sqsubseteq \ \ldots$ (with $m > n$)
  "*There can't be at-least m and at-most n fillers for the property r. The subsumee is equivalent to* $\bot$ *which is subsumed by everything.*"
- $\ldots \sqsubseteq (\leq n\ r) \sqcup (\geq m\ r)$ (with $m \leq n + 1$)
  "*There are always either less than n or more than m fillers for the property r. The subsumer therefore is equivalent to* $\top$ *which subsumes everything.*"[1]
- $(\leq n\ r) \sqsubseteq (\leq m\ r)$ (with $m \geq n$)
  "*At most n fillers for r is subsumed by at most m fillers for r.*"
- $(\geq m\ r) \sqsubseteq (\geq n\ r)$ (with $m \geq n$)
  "*At least m fillers for r is subsumed by at least n fillers for r.*"[2]

Another extension of the reasoning procedure is concerned with the combination of existential quantifications and at most restrictions. Moreover, a $(\leq\ 1\ r)$ restriction forces a property r to have at most one r-successor. In case of additional existential quantification this requires to merge all existing successor nodes to one single node. Since we do

[1]This kind of clash/explanation can only occur in the $\mathcal{ALC}$ language family.

[2]Within OWL Lite ($n = 0$) this case does not occur (rhs will reduce to $\top$ beforehand).

not allow for full negation such a merge can only locally occur on lhs. Consider the query $\exists r.A \sqcap \exists r.B \sqcap (\leq 1\ r) \sqsubseteq \exists r.(A \sqcap B)$. As a result of the at most restriction the r-successor node will be explained as follows: "*Since there has to be at least one filler for each of the types A and B and at most one filler for r on lhs this filler has to be of type* $(A \sqcap B)$".

An analogous explanation is needed for functional properties even without an explicit at most restriction. E.g. $\exists r.A \sqcap \exists r.B \sqsubseteq \exists r.(A \sqcap B)$ holds in case of a functional property r. The corresponding explanation would state: "*Since r is functional there is at most one filler which has to be of type* $(A \sqcap B)$".

Property Hierarchies. OWL allows to specify a sub-properties which result in property hierarchies. Within such a property hierarchy each property filler is also a filler of all super properties. For example, when having *son* as a sub-property of *child*, each existing son will be a child at the same time. However, the effect of restriction over a property in a tableau proof is the reverse — from a property to its sub-properties. Thus, quantitative restrictions of a property like domain and range as well as filler types also apply to all sub-properties. Consider a sub-property p of r ($p \sqsubseteq r$) and an expression $(\geq 1\ p) \sqcap \forall r.A$. Each p-successor node will get an additional explanation stating that "*All fillers of r are restricted to be of type A. Since p is a sub-property of r this restriction also applies to p*."

Cardinality restrictions and merging require a likewise handling. As with "direct" cardinality restrictions we have to take the side of each expression into account and therefore need to distinguish between four different types of clashes.

- $(\leq n\ r) \sqcap (\geq m\ p) \sqsubseteq \ \ldots$
"*There can't be at most n fillers for the property r and at least m fillers for the sub-role p at the same time. The subsumee is equivalent to BOTTOM which is subsumed by everything*."
- $\ldots \sqsubseteq (\leq n\ p) \sqcup (\geq m\ r)$ for $m = n + 1$ and $p \sqsubseteq r$
"*There are always either less than n fillers of sub-property p or more than m fillers for the property r. The subsumer therefore is equivalent to TOP which subsumes everything*."[3]
- $(\leq n\ r) \sqsubseteq (\leq m\ p)$
"*At most n fillers for r is subsumed by at most m fillers for sub-property p*."
- $(\geq m\ p) \sqsubseteq (\geq n\ r)$
"*At least m fillers for sub-property p is subsumed by at least n fillers for r*.[4]"

Domain and Range Restrictions. Global domain and range restrictions of a property become proof relevant as soon as a tableau rule generates a successor node for that property. The given range or domain restriction will then be added to the successor resp. predecessor node by the tableau algorithm. Such a restriction may consist of an arbitrary expression of the underlying language. As a consequence, in case of a resulting clash due to a domain or range restriction one of the previously mentioned explanations will apply. In addition, the source of the domain or range expression has to be explained. Accordingly, we need additional explanation steps for domain and range restrictions. Hence, whenever a property r has a domain or range restriction an additional explanation statement will be given just after a generating r-successor rule. As an example, consider the following query $\exists r.A \sqsubseteq \exists r.B$ and a range restriction B on the property r. The two existential quantifications are then explained as described before: "*For the property r we have to check whether* $A \sqsubseteq B$". Because of the relevant range restriction an additional explanation step states "*Since the property r has a range restriction on B the subsumption evolves to* $(A \sqcap B) \sqsubseteq B$".

Transitive Properties. Restrictions on transitive properties ($\forall r.C$ with r transitive) have to be propagated to all descendants referenced over a sequence of r-successors. In the tableau algorithm this is achieved by adding the qualifier C as well as the whole construct $\forall r.C$ to all direct successors of the node containing the restriction $\forall r.C$. In order to ensure termination an apropriate blocking strategy is required.

With respect to explaining the propergation of restrictions result in additional expressions within successive sub-explanations which do not obviously follow from the previous step. In order not to let sub-proofs change in an unobvious way we add an additional explanation statement is this situations. As an example, the qualified universal quantification from above would result in the following additional explanation: "*Since r is a transitive property* $\forall r.C$ *also has to hold for all its successors*".

Optimizations

To generate concise and simple explanations we have implemented several optimizations which condense the explanation in specific situations called *filtering* and *mode switching*.

Filtering is a simple method for pruning disjunction on rhs with help of a structural comparison. Consider the subsumption $A \sqcap B \sqcap C \sqsubseteq A \sqcap C$. The standard approach would split up the explanation into two sub-subsumptions according to the internal disjunction on rhs. Very likely this is unnecessary because this subsumption obviously holds, since the subsumee is a direct specialization of the subsumer. It is our opinion, this is a commonly agreed and intuitive conclusion which naturally follows from the relationship between conjunction and specialization. Therefore our approach structurally compares the lhs and rhs after lazy unfolding and prior to further processing. If the rhs is a syntactical subset of the lhs an obvious subsumption is found and explained accordingly.[5]

Another optimization method is called mode switching and applies to situations where at some stage of tableau processing either the rhs or lhs is unsatisfiable on its own (i.e.

[3] This kind of clash/explanation can only occur in the $\mathcal{ALC}$ language family.

[4] Within OWL Lite ($n = 0$) this case does not occur (rhs will reduce to $\top$ beforehand).

[5] A similar technique, called normalizing and naming, is found in state of the art DL reasoning systems.

independently from each other). This corresponds to either the subsumee being equivalent to $\bot$ or the subsumer being equivalent to $\top$ on explanation side. In such a case our explainer will switch to either unsatisfiability or tautology explaining while disregarding the other side. Since our explanations are generated on-the-fly this requires to check each side concerning unsatisfiability in advance for which we use an optimized DL reasoner called Racer (Haarslev & Möller 2001).

An additional benefit of the optional Racer connection is the increased performance when using the dual-reasoner architecture as proposed in (Kwong 2004). This optimization feature uses Racer to determine the effectively clash relevant expressions within each side. This will prune the tableau of the explainer by leaving out non relevant node expansions.

Explainer Prototype MEX

Our explainer MEX (Liebig & Halfmann 2005) is implemented in Lisp and capable of explaining subsumption within the language described above. Its syntax for defining concepts and roles follows the KRSS standard (Patel-Schneider & Swartout 1993), which easily allows to transfer existing ontologies from other systems. MEX is equipped with an optional Racer link-up. Using Racer leads to optimized explanations (with mode switching) and an increased performance. As mentioned before, MEX creates all explanations in parallel to the tableau tree generation. This is a disatvantage with respect to different detailed explanations as well as explanation length. We plan to redesign the systems architecture in order to support a two step procedure: create explainer tableau first and generate explanation afterwards.

Combining Authoring with Explaining

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 we made our explainer accessible to our interactive ontology authoring tool ONTOTRACK (Liebig & Noppens 2005) via its plug-in interface.

ONTOTRACK is a new browsing and editing "in one view" ontology authoring tool for OWL Lite$^-$ hat combines a sophisticated graphical layout with mouse-enabled editing features and instant reasoning feedback optimized for efficient navigation and manipulation of large ontologies. An ontology is drawn as a directed acyclic graph with classes as nodes and the subsumption relationship as edges either in top-down, left-right, bottom-up, or right-left orientation. To invoke the on-demand explanation facility, the user only needs to hover the mouse pointer over the edge representing the relevant subsumption relationship and to select the "Explain" button within the context menu.

ONTOTRACK then converts the internal ontology model into a standard syntax (KRSS) and sends it via TCP together with an explanation request to the explaining component. An explanation is generated and retransmitted as an XML serialization back to ONTOTRACK. This enables the ONTOTRACK plug-in to display the explanation in a graphical way as an expandable tree list.

Figure 3 shows the ONTOTRACK application together with an explanation for class *PersonWithChild* subsuming *ParentWithPorsche*.

Related Work

An early work on explaining TBox reasoning was an extension of the Classic system using a structural inference algorithm for a less expressive language (McGuinness 1996). An approach for explaining subsumption by computing an interpolation of an intermediate concept in-between subsumer and subsumee for $\mathcal{ALC}$ is given in (Schlobach 2004). The most related work seems to be the dual-reasoner approach (Kwong 2004) which is also based on a tableau algorithm. Two distantly related approaches try to explain unsatisfiability by using a DL reasoner as an external service without taking advantage of the internal reasoning process (so-called "black box" approach). One uses a heuristic which tries to identify commonly made modeling mistakes (Wang *et al.* 2005). The other is a diagnosis approach in order to find helpful debugging cues (Parsia, Sirin, & Kalyanpur 2005).

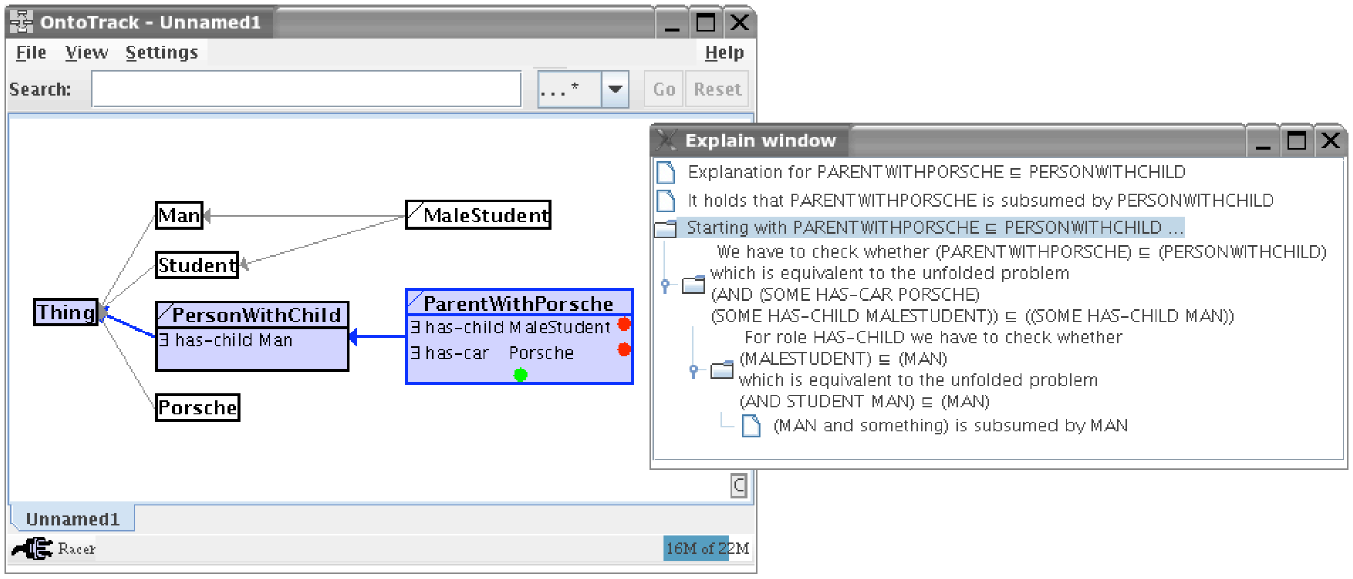
Discussion and Outlook

Our approach of explaining subsumption covers a fairly expressive DL. To our knowledge, the system we built is the first tableau-based explainer capable to explain such an expressive language. In order to deal with all of OWL Lite requires support for general concept inclusion (GCI), inverse roles as well as multiple definitions for a single class identifier. In addition, explaining subsumption within OWL DL requires to explain nominals (ABox individuals) as well as unrestricted cardinality constraints. Most of the latter is either costly in terms of processing as well as explanation length (because of non-determinism) or currently not known explainable in a comprehensible way at all.

Our short term goal is to enhance the ONTOTRACK integration to allow for more sophisticated features like highlighting of referenced descriptions (classes or properties) or special views which show only those classes which are relevant within a certain subsumption query. We also aim to enhance the textual presentation by providing different detailed alternatives which can be achieved by hiding non-relevant expressions within queries.

Future work is concerned with more expressive languages as well as additional explanation questions. Note that explaining unsatisfiability can be considered as a special case of our approach. Explaining equivalence between classes or "everything" that can be inferred about a class my also be of interest by an end-user. Another related idea is to explain why a subsumption between two classes does not hold. Here, a counter-example could be distilled out of a tableau proof.

Other "non-standard inference" services also have a great potential in supporting the ontology authoring process. These include concept matching, rewriting or approximation, as well as the least common subsumer or the most specific concept (Brandt & Turhan 2001). However, they are subject of ongoing research and it is currently unknown



Explanation for PARENTWITHPORSCHE ⊑ PERSONWITHCHILD
It holds that PARENTWITHPORSCHE is subsumed by PERSONWITHCHILD
Starting with PARENTWITHPORSCHE ⊑ PERSONWITHCHILD ...
We have to check whether (PARENTWITHPORSCHE) ⊑ (PERSONWITHCHILD)
which is equivalent to the unfolded problem
(AND (SOME HAS-CAR PORSCHE)
(SOME HAS-CHILD MALESTUDENT)) ⊑ ((SOME HAS-CHILD MAN))
For role HAS-CHILD we have to check whether
(MALESTUDENT) ⊑ (MAN)
which is equivalent to the unfolded problem
(AND STUDENT MAN) ⊑ (MAN)
(MAN and something) is subsumed by MAN

Figure 3: Screenshot of ONTOTRACK displaying a subsumption explanation.

if they are extendible to OWL expressiveness (Horrocks 2005). If so, they also could be used to simplify concept expressions (with help of rewriting or approximation) in order to reduce the complexity of explanation queries for example. Concerning non-subsumption, unification modulo subsumption could be utilized to compute a measure of "difference" between two classes. However, a premise to this is that those inferences can be explained on their part.

References

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P., eds. 2003. *The Description Logic Handbook*. Cambridge Univesity Press.

Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P.; and Stein, L. A. 2004. OWL Web Ontology Language Reference. W3C Recommendation. `http://www.w3.org/TR/owl-ref/`.

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

Brandt, S., and Turhan, A.-Y. 2001. Non-standard Inferences in Description Logics—what does it buy me? In *Proc. of the 2001 International Workshop on Description Logics (DL01)*.

Fiedland, N.; Allen, P.; Witbrock, M.; Matthews, G.; Salay, N.; Miraglia, P.; Angele, J.; Stab, S.; Israel, D.; Chaudhri, V.; Porter, B.; Barker, K.; and Clark, P. 2004. Towards a Quantitative, Plattform-Independent Analysis of Knowledge Systems. In *Proc. of the International Conf. on Principles of Knowledge Representation and Reasoning*, 507–514.

Haarslev, V., and Möller, R. 2001. Description of the RACER System and its Applications. In *Proc. ot the Int. Workshop on Description Logics (DL01)*.

Horrocks, I., and Sattler, U. 2005. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*. To appear.

Horrocks, I.; Patel-Schneider, P. F.; and van Harmelen, F. 2003. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics* 1(1):7–26.

Horrocks, I. 2005. Applications of Description Logics: State of the Art and Research Challenges. In *Proc. of the 13th Int. Conf. on Conceptual Structures (ICCS'05)*. to appear.

Kalyanpur, A.; Parsia, B.; and Hendler, J. 2005. A Tool for Working with Web Ontologies. *International Journal on Semantic Web and Information Systems* 1(1):36–49.

Klyne, G., and Caroll, J. 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation. `http://www.w3.org/TR/rdf-concepts/`.

Kwong, F. 2004. Explaining Description Logic Reasoning. In *Proc. of the 2004 International Workshop on Description Logics (DL04)*, 210.

Liebig, T., and Halfmann, M. 2005. A Tableau-Based Explainer for DL Subsumption. In *Proc. of the Int. Conference on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux2005)*. to appear.

Liebig, T., and Noppens, O. 2004. ONTOTRACK: Combining Browsing and Editing with Reasoning and Explaining for OWL Lite Ontologies. In *Proc. of the International Semantic Web Conf. (ISWC 2004)*, 244–258. Hiroshima, Japan: Springer Verlag.

Liebig, T., and Noppens, O. 2005. ONTOTRACK: A Semantic Approach for Ontology Authoring. *Journal of Web Semantics* 3(2).

McGuinness, D. 1996. *Explaining Reasoning in Description Logics*. Ph.D. Dissertation, Rutgers University.

Parsia, B., and Sirin, E. 2004. Pellet: An OWL DL Reasoner. In *Poster Proc. of the 3rd International Semantic Web Conference (ISWC)*, 57–58.

Parsia, B.; Sirin, E.; and Kalyanpur, A. 2005. Debugging OWL Ontologies. In *The International World Wide Web Conf. (WWW2005)*.

Patel-Schneider, P., and Swartout, B. 1993. Description-Logic Knowledge Representation System Specification. Working Version (Draft).

Rector, A.; Drummond, N.; Horridge, M.; Rogers, J.; Knublauch, H.; Stevens, R.; Wang, H.; and Wroe, C. 2004. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In *Proc. of the European Conf. on Knowledge Acquistion (EKAW04)*, 63–81.

Schlobach, S. 2004. Explaining Subsumption by Optimal Interpolation. In *Proc. of the European Conf. of Logics in Artificial Intelligence*, 413–425.

Wang, H.; Horridge, M.; Rector, A.; Drummond, N.; and Seidenberg, J. 2005. A Heuristic Approach to Explain the Inconsistency in OWL Ontologies. In *Proc. of the Intl. Protégé Conference*.