Understanding Entailments in OWL

Matthew Horridge[1] and Johannes Bauer[1] and Bijan Parsia[1] and Ulrike Sattler[1]

The University of Manchester
Email: matthew.horridge@cs.man.ac.uk

Abstract. This paper describes the explanation in OWL landscape. In recent years there has been huge progress, both in theory and implementation, in the area of explaining the causes of entailments in OWL ontologies. This paper charts the course of explanation in OWL and then looks at ways in which user understanding of ontologies might be further improved. Specifically, the use of fine-grained justifications, augmentation of justifications with lemmas, and the provision of browseable models are discussed as methods of further improving user understanding in the context of ontologies and entailments.

1 Introduction

In 2003, as the Web Ontology Language OWL [12] was on the verge of becoming a standard, one of the first OWL ontology editors, Protégé-OWL [8] was released. This was in addition to OILed [1], which had built in support for saving DAML+OIL [4] ontologies using the OWL vocabulary, and had been released a few years earlier. Both of these editors shared the capability of being able to "connect" to description logic reasoners such as Pellet [14], FaCT++ [15] and Racer [2], in order to perform standard description logic reasoning services, such as satisfiability checking and subsumption testing, on the ontologies being edited. Many users were enticed into using these reasoning services when building their ontologies, because they saw the use of them as a checking or compilation step during the course of ontology development. Indeed, it was typically the case that users found it useful when modelling errors were identified through the use of reasoning, in particular when unsatisfiable classes were identified and highlighted in the UIs of these editors.

Both OILed and Protégé had little in the way of support for debugging ontologies. It was not possible to obtain explanations as to why classes were unsatisfiable. The best debugging support that was available at the time was the practice of painting unsatisfiable classes in red, which made them relatively easy to spot. In turn, this allowed users to manually "trace" through the ontology so as to spot patterns and locate the part of the ontology that they should concentrate on when trying to understand the reasons for, and get rid of, the unsatisfiable classes. If a class was unsatisfiable, users would usually look to see if any of the super-classes were unsatisfiable and then concentrate on those. Likewise, they would check to see if the fillers of any existential restrictions were unsatisfiable, and if so, navigate to these, and then attempt to spot why they were unsatisfiable.

Having narrowed down the axioms on which they should concentrate on, many users would then start to 'rip out' axioms from the ontology. Most notably people would remove disjoint classes axioms, in an attempt to rid an ontology of any unsatisfiable classes.

All in all, debugging an ontology that contained unsatisfiable classes was a wretched and error prone process. So much so that in some cases, users were afraid to use a reasoner to check their ontologies. At worst, for OWL, users would switch to a different or legacy knowledge representation language such as Frames [11], as they perceived these languages as being easier to understand and use.

Since the early days of OWL, there have been huge advancements in the debugging facilities provided by ontology development environments. Most notably, the ontology editor Swoop [7], from the MIND lab at the University of Maryland provided practical implementations of explanation services to support debugging, making it possible to obtain an explanation for any entailment that was exposed through the user interface. Since then debugging support was steadily incorporated into other tools and ontology editors. Indeed, today it is arguable that no respectable ontology editor should be without the ability to provide explanations for why entailments hold to end users .

The purpose of this paper is to review work in the area of debugging and explanation for OWL ontologies, take a look at debugging support in the mainstream ontology editors, peek at current state of the art work on explanation, and finally speculate on possible future directions for explanation and debugging support.

2 Preliminaries

2.1 OWL-DL and OWL 2

OWL-DL is a flavour of OWL that corresponds to the description logic $\mathcal{SHOIN}(\mathcal{D})$. OWL 2, which corresponds to the description logic $\mathcal{SROIQ}(\mathcal{D})$, is the latest version of OWL that enhances OWL-DL to make it more expressive by adding new kinds of class constructors and axioms. Herein, OWL is now used to refer to OWL 2. It is assumed that the reader is familiar with the various OWL class constructors and axioms. For an in-depth review of OWL the interested reader is referred to [5]. What follows is a recap on interpretations and models.

The semantics of OWL is given by interpretations. Interpretations explicate the relationship between syntax and semantics. An interpretation, $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, is tuple that consists of a non-empty interpretation domain, $\Delta^{\mathcal{I}}$, and an interpretation function, $\cdot^{\mathcal{I}}$. The interpretation function maps each class name A into a subset $A^{\mathcal{I}}$ of the domain, each property name P into a subset $R^{\mathcal{I}}$ of a binary relation over the domain, and each individual name a into to an object $a^{\mathcal{I}}$ in the domain. The interpretation function is extended to deal with complex class descriptions and axioms. If an interpretation satisfies every axiom in an ontology $\mathcal{O}$ then the interpretation is said to be a *model* of $\mathcal{O}$.

Example Consider a very small ontology containing a single axiom $\mathcal{O} = \{Car \sqsubseteq Vehicle\}$. Let $\Delta^{\mathcal{I}} = \{x_0\}$ (that is, the interpretation domain $\Delta^{\mathcal{I}}$contains one object x_0) and let Car and $Vehicle$ be interpreted as follows: $Car^{\mathcal{I}} = \{x_0\}$, $Vehicle = \emptyset$. This particular interpretation is *not* a model of $\mathcal{O}$ because x_0 is not in the interpretation of $Vehicle$ and hence the interpretation does not satisfy the axiom $Car \sqsubseteq Vehicle$. However, the interpretation $Car^{\mathcal{I}} = \{x_0\}$, $Vehicle = \{x_0\}$ *is* a model of $\mathcal{O}$ because it satisfies all axioms in $\mathcal{O}$.

2.2 Terminology

In what follows terminology that is related to the field of explanation and debugging is reviewed. Ontologies that are incoherent, because they contain unsatisfiable classes, or ontologies that are inconsistent because they do not have any models, generally arise as the result of modelling errors. Entailments such as classes being unsatisfiable, or ontologies being inconsistent are usually viewed as being undesirable entailments.

Signature The signature of an ontology $\mathcal{O}$ is the set of class, property and individual names that are used in axioms in $\mathcal{O}$. For example, consider $\mathcal{O} = \{A \sqsubseteq \exists R.C, B \sqsubseteq \forall S.D\}$, where the signature of $\mathcal{O}$ is $\{A, R, C, B, S, D\}$.

Unsatisfiable Classes A class is unsatisfiable (with respect to an ontology) if it cannot possibly have any instances in any model of that ontology. More precisely, a class is unsatisfiable if and only if it is interpreted as the empty set in all models. Since unsatisfiable classes are always interpreted as the empty set, and the empty set is a subset of every set, then an unsatisfiable class is a subclass of every class. In description logic notation, $C \sqsubseteq \bot$ means that C is unsatisfiable.

Incoherent Ontologies In the context of debugging and explanation, an incoherent ontology is an ontology that contains at least one unsatisfiable class. More precisely, an ontology $\mathcal{O}$ is incoherent if and only if $\mathcal{O} \models C \sqsubseteq \bot$ for at least one class name C in the signature of $\mathcal{O}$.

Inconsistent Ontologies An ontology $\mathcal{O}$ is inconsistent if and only if $\mathcal{O}$ does not have any model. An inconsistent ontology entails $\top \sqsubseteq \bot$. It should be noted that an ontology that just contains unsatisfiable classes (other than $\top$) is not inconsistent.

Entailment We write $\mathcal{O} \models \eta$ if all models of $\mathcal{O}$ also satisfy η. In this case "$\mathcal{O}$ entails η", and we also say that η is an entailment in $\mathcal{O}$. A standard description logic reasoner test each class for (un)satisfiability, each ontology for consistency and can answer various entailment queries.

Justifications Justifications are a type of explanation for entailments in ontologies.

Let $\mathcal{O}$ be an ontology and η any arbitrary entailment, such that $\mathcal{O} \models \eta$ ($\mathcal{O}$ entails η). Then $\mathcal{J}$ is a justification for η in $\mathcal{O}$ if $\mathcal{J} \subseteq \mathcal{O}$, $\mathcal{J} \models \eta$ and for any $\mathcal{J}' \subset \mathcal{J}$ $\mathcal{J}' \not\models \eta$. Intuitively, a justification for an entailment in an ontology, is a minimal subset of the ontology that is sufficient for the entailment in question to hold. A justification is minimal in the sense that for any proper subset of the justification, the entailment in question does not hold.

Root and Derived Unsatisfiable classes A class C that is unsatisfiable with respect to an ontology $\mathcal{O}$ is a *derived* unsatisfiable class if there exists a justification $\mathcal{J}$ for $\mathcal{O} \models C \sqsubseteq \bot$, and a justification $\mathcal{J}'$ for $\mathcal{O} \models D \sqsubseteq \bot$ such that $\mathcal{J} \supset \mathcal{J}'$. A class that is not a *derived* unsatisfiable class is known as a *root* unsatisfiable class. An unsatisfiable class, all of whose justifications are supersets of at least one justification for another unsatisfiable class is known as a *pure derived* unsatisfiable class.

In a nutshell, derived unsatisfiable classes have justifications that are supersets of the justifications for *some other* unsatisfiable class in the same ontology. If an unsatisfiable class does not have any justifications that are supersets of justifications for some other unsatisfiable class then the class is a root unsatisfiable class. The significance of this is that a user should aim to repair root unsatisfiable classes before repairing all other unsatisfiable classes because some or all derived unsatisfiable classes may also be repaired in the process of doing this.

3 User facing tools

The first tool to bring sound and complete explanation generation facilities to end users was Swoop. The comprehensive debugging and repair facilities in this tool go a long way in addressing many of the issues identified in the previous section. Given an incoherent ontology, a user is able to determine the root/derived unsatisfiable classes so that they know which classes to concentrate on fixing. They can then browse justifications for these unsatisifable classes in order to attempt to understand why the classes are unsatisfiable. Finally they can use the repair tool in Swoop to suggest a repair plan for the ontology that will ultimately result in all unsatisfiable classes turning satisfiable.

An example justification, for $DNA \equiv \bot$, as displayed in Swoop, is shown in Figure 1. As can be seen, Swoop presents justifications as ordered indented lists of axioms. However, these are only based on heuristics—there hasn't been any research into an optimal ordering of axioms. For a given axiom, the right hand side (RHS) of the axiom is established and then examined so that any names appearing in the RHS signature are used to indicate which axioms should immediately follow and be indented.

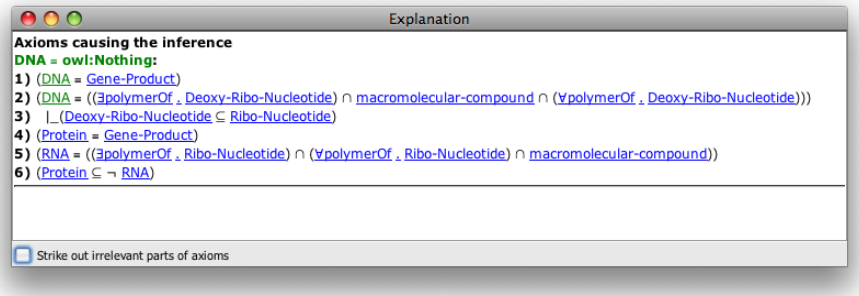

Fig. 1. An example justification as presented in the ontology editor Swoop

4 User centred tasks

Broadly speaking, explanation tools in ontology development environments should ideally satisfy the following use cases. Each use case is a task that a user of a tool that generated explanations might want to accomplish.

1. **Understanding entailments**—A user browsing an ontology notices an entailment and opportunistically decides to obtain an explanation for the entailment in order to get a feel as to why the entailment holds.
2. **Debugging and repair**—A user is faced with an incoherent ontology, or an ontology that contains some other kind of undesirable entailment, and they need to determine the causes of the entailments in order to generate a repair plan.
3. **Ontology comprehension**—A user is faced with an ontology that they haven't seen before. In order to get a better picture of the ontology they use various metrics such as the number of entailments, the average number of justifications for an entailment and so on. This helps them to build up an image of how complex the ontology is in terms of expressivity. Is also provides them with more information if they need to decide whether they like the ontology or not.
4. **Understanding justifications**—Once a justification for an entailment in an ontology has been obtained, a user wants to understand the justification better. For example, they would like to know what entailments arise from the justification and which axioms within the justification itself cause the entailments to hold.

Over the past few years, ontology development environments have gone from no support for these tasks through to respectable support for Tasks 1 and 2. Indeed, since Swoop set the bar for explanation and repair facilities in ontology development environments, other tools have slowly begun to offer similar facilities. However, taking tools in this area into consideration, there are still some holes that need to be filled. In particular, it is arguable that the use of

explanations and justifications for ontology comprehension, and the ability to gain more insight and understanding into the justifications themselves are still under-supported tasks. The remainder of the paper discusses work in progress in the field of explanation and understanding of OWL ontologies.

5 Fine-grained Justifications

Due to the typical construction of rich ontologies, and the way in which ontology development environments display and make it easy to edit axioms, it is frequently the case that axioms can be rather long. Hence, justifications can contain "long" axioms, where only *part* of the axioms are required for the entailment in question to hold. In many cases, these parts can obfuscate the true reasons as to why an entailment holds. Justifications that contain long axioms could also result in information being unnecessarily lost when repairing an ontology through deleting axioms, because it isn't clear which parts of the axioms contribute to the entailment explained by the justification.

The ontology editor Swoop uses *heuristics* to strike out class, property and individual names that are superfluous to an entailment. An example of strikeout in action is shown in Figure 2, where the names *process*, *contains* and *isomers* have been struck out. This has the advantage of focusing the users attention on the relevant parts of the justification.

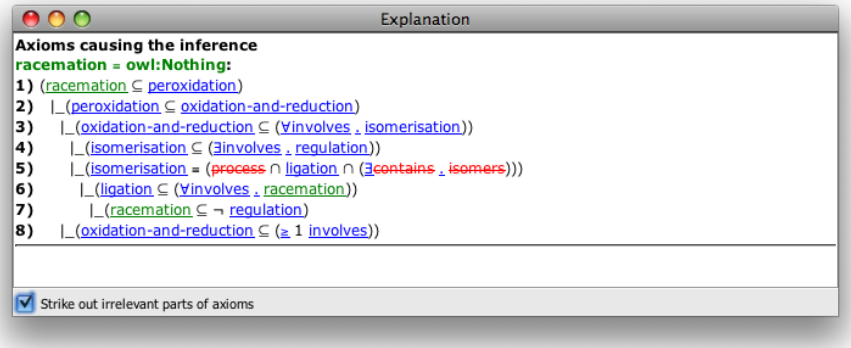

Fig. 2. An example justification

Justifications that contain axioms that do not contain any redundant parts have been known as *fine-grained justifications* [9] or *precise justifications* [6]. For lack of a formal definition of fine-grained justifications, various implementers have used different ad-hoc approaches to computing them. As well as the heuristic based strikeout feature used in Swoop [6], other examples of computing these kinds of justifications include the repair tool developed as part of Lam's PhD

thesis [10], and some syntactic generalisation techniques proposed by Schlobach [13]. A problem with prior approaches is that they essentially defined fine-grained justifications in an operational sense by specifying how to compute them. This has made it difficult to pin down what a fine-grained justification is. However, recent theoretical work by the authors [3][1] has changed this situation by providing a formal definition of fine-grained justifications.

We have defined *laconic justifications*, which informally, are justifications that only consist of axioms that do not contain any superfluous parts and whose parts are as small and as weak as possible. *Precise justifications* can be derived from laconic justifications, and contain axioms that are as small, flat and as weak as possible. Laconic justifications are aimed at improving understanding, while precise justifications are aimed at guiding the construction of a semantically minimal repair (see [3] for further details).

This work on precise justifications resulted in some surprising results when experiments were performed on several publicly available ontologies. For example, for some ontologies, it was found that for a given entailment, the number of laconic justifications were fewer in number than the number of regular justifications. Examples were also found where regular justifications *masked* further laconic justifications. Full details are available in [3], an example of masking is shown here:

Example One of the main issues with regular justifications is that for a given entailment they can *mask* other justifications. Consider the following ontology $\mathcal{O} = \{C \sqsubseteq B \sqcap \neg B \sqcap D, C \sqsubseteq \neg D\}$ which entails $C \sqsubseteq \bot$ (C is unsatisfiable). A justification for C being unsatisfiable would *never* include $C \sqsubseteq \neg D$. This is clearly undesirable as it could hamper the possibility of choosing the correct repair plan to remove this undesirable entailment. A real example of such masking was found in the DOLCE ontology. The entailment `quale` $\sqsubseteq$ `region` has a single justification: {`quale` $\equiv$ `region` $\sqcap \exists$ `atomic-part-of.region`}. However, computing laconic justifications for this entailment reveals that there are further justifications that are masked by this regular justification. There are three laconic justifications, the first being {`quale` $\sqsubseteq$ `region`}, which is directly obtained as a weaker form of the regular justification. This first laconic justification could be identified in Swoop using the strike out feature (The conjunct $\exists$ `atomic-part-of.region` would be struck out). More interestingly, Figure 3 shows two additional laconic justifications.

5.1 Automatic Lemma Generation

While justifications have proved to be incredibly useful for end users when debugging ontologies, preliminary experimental evidence suggests that, in many cases, even with justifications in hand, users can still find it difficult to understand the causes of entailments. The exact reasons for this are unknown. However, in the course of observing users who are tying to understand justifications, it has been

[1] Accepted as a paper in the research track at ISWC this year.

$\texttt{quale} \sqsubseteq \exists \texttt{atomic-part-of.region}$	$\texttt{quale} \sqsubseteq \texttt{atomic-part-of.region}$
$\texttt{atomic-part-of} \sqsubseteq \texttt{part-of}$	$\texttt{atomic-part-of} \sqsubseteq \texttt{atomic-part}^{-}$
$\texttt{part-of} \sqsubseteq \texttt{part}^{-}$	$\texttt{atomic-part} \sqsubseteq \texttt{part}$
$\texttt{region} \sqsubseteq \forall \texttt{part.region}$	$\texttt{region} \sqsubseteq \forall \texttt{part.region}$

Fig. 3. Masked justifications from the DOLCE ontology

noted that there are certain justifications that seem difficult for most users to understand. The authors hypothesise that these justifications contain non-obvious ("hidden") entailments, and in order to understand the whole justification, a user must spot these non-obvious entailments.

An example of such a case is shown in Figure 4 which is taken from an ontology about movies that was posted to the Protégé mailing list [2]. In this example, the justification for $Person \sqsubseteq Movie$ also entails that the class $Movie$ is equivalent to $Thing$, and hence every class is a subclass of $Movie$. However, this isn't explicit in the justification, and for most people this is far from obvious, yet it is critical to realise that this entailment holds in order to understand the explanation.

$$Person \sqsubseteq \top \tag{1}$$
$$ParentalAdvSuggested \equiv \forall hasViolenceLevel.Medium \tag{2}$$
$$hasViolenceLevel\ domain\ Movie \tag{3}$$
$$ParentalAdvSuggested \sqsubseteq CertificationCatMovie \tag{4}$$
$$CertificationCatMovie \sqsubseteq Movie \tag{5}$$

Fig. 4. A justification for $Person \sqsubseteq Movie$. This is an example of a justification that seems to be difficult for most users to understand.

One possible solution is to augment justifications with automatically generated lemmas. These lemmas can help to bridge the gap in understanding, highlighting the non-obvious entailments that are required in order to understand a justification. What lemmas should be used in what context is presently unknown. In order to come up with a recipe for augmenting justifications with lemmas, an investigation is under way to determine the factors that make justifications difficult to understand. It is hoped that this will result in a model for the complexity of understanding a justification. With this model in hand, it should then be possible to start to identify whether or not a justification is difficult to understand and decide which parts of the justification should be bridged with lemmas. It is expected that lemmas will be useful for an immediate

[2] http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/22321/focus=22370

overview of a justification and allow a user to "drill down" into the justification should they require more understanding of the entailment or the need to carry out a repair of the ontology. As an example, some of the axioms in Figure 4 could be replaced as follows: axiom 2 could be rewritten to make it more laconic and to show a negated existential restriction so that it becomes $\neg\exists hasViolenceLevel.\top \sqsubseteq ParentalAdvisorySuggested$, axiom 4 (the property domain axiom) could be rewritten to make the existential implication explicit so that it becomes $\exists hasViolenceLevel\top \sqsubseteq Movie$, and axioms 4 and 5 could be replaced with one axiom $ParentalAdvSuggested \sqsubseteq Movie$.

6 Models—SuperModel

Another possible strategy for users attempting to get a grasp on an ontology and the kinds of entailments that hold in it is for them to think about *models*. Many people find the notion of models very natural, and are perfectly happy to browse diagrams of blobs connected by lines. Figure 5 shows a screenshot of "SuperModel", which is a new prototype tool for browsing models of class descriptions. The tool allows users to select a class description (from the hierarchy on the left hand side) and then shows a model of the class description, rooted at the "rootIndividual" instance, on the right hand side. Arcs labelled with the names of properties represent relationships to other individuals. Users of tools such as Protégé tend to be not unfamiliar with such visualisations, because they have seen components that can display similar diagrams of an ontology. However, this is where the similarity with other visualisation tools ends. Because SuperModel displays models generated by description logic reasoners, it is capable of showing *entailed relationships* between individuals and *entailed types* of individuals. By clicking on a node in the graph, a user is able to view the classes that an individual, which is represented by the node, belongs to.

In the example in Figure 5 the class $Margherita$ has been selected from the pizza ontology. SuperModel shows an example of a model for $Margherita$. It is easy to see that an individual that is a $Margherita$ (pizza) has two $hasTopping$ relationships and that one of these $hasTopping$ successors has a $hasCountryOfOrigin$ successor to an individual that is $Italy$. SuperModel allows the user to gradually expand the model in order to fully explore it.

One possible use of SuperModel is that it could be used to given an indication of the reasons for non-subsumption. A frequent question on mailing lists is "Why isn't X inferred to be a subclass of Y?". In such situations, SuperModel is able to generate example models that are "counter-models" that show how an individual could be an instance of the subclass (X) and not an instance of the superclass Y. At this stage, we are investigating whether or not the use of counter-models as an aid in understanding the reasons for non-subsumption is of benefit to a typical ontology modeller.

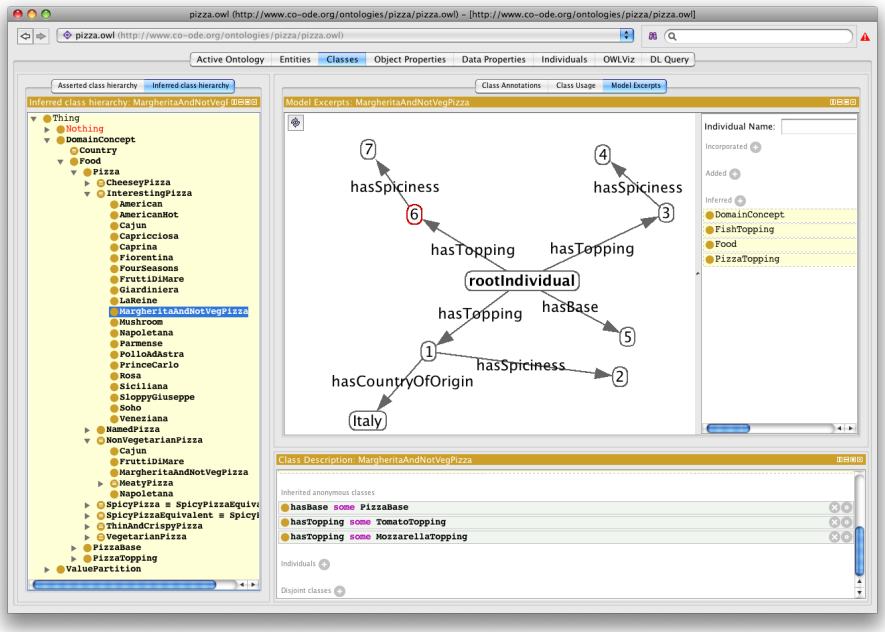

Fig. 5. A screenshot of the SuperModel plugin in Protégé

7 Conclusions

Over the past few years since OWL became a standard, services and tools for generating explanations of entailments have come from nothing to being more than respectable. In particular, the ability to generate *justifications* for entailments is now seen as a key inference service that is required for the development of ontologies. However, anecdotal evidence suggests that certain types of justifications can be very hard if not impossible for a broad range of users to understand. Work to remedy this situation includes identifying superfluous parts of axioms using so-called *laconic* justifications, experimenting with augmenting justifications with lemmas, and the use of models for improving ontology understanding and comprehension.

References

1. Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. OilEd: A Reasonable ontology editor for the semantic web. In *Proc. of the Joint German/Austrian Conf. on Artificial Intelligence (KI 2001)*, number 2174 in Lecture Notes in Artificial Intelligence, pages 396–408. Springer, 2001.
2. Volker Haarslev and Ralf Moller. Racer system description. In *International Joint Conference on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes In Computer Science*, pages 701–705, 2001.

3. Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in owl. In *ISWC 08 The International Semantic Web Conference 2008, Karlsruhe, Germany*, 2008.
4. Ian Horrocks. DAML+OIL: a reason-able web ontology language. In *Proc. of EDBT 2002*, number 2287 in Lecture Notes in Computer Science, pages 2–13. Springer, March 2002.
5. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.
6. Aditya Kalyanpur, Bijan Parsia, and Bernardo Cuenca Grau. Beyond asserted axioms: Fine-grain justifications for owl-dl entailments. In *DL 2006, Lake District, U.K.*, 2006.
7. Aditya Kalyanpur, Bijan Parsia, and James Hendler. A tool for working with web ontologies. In *International Journal on Semantic Web and Information Systems*, volume 1, Jan - Mar 2005.
8. Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. The protege owl plugin: An open development environment for semantic web applications. In *ISWC 04 The International Semantic Web Conference 2004, Hiroshima, Japan*, 2004.
9. Joey Sik Chun Lam, Derek H. Sleeman, Jeff Z. Pan, and Wamberto Weber Vasconcelos. A fine-grained approach to resolving unsatisfiable ontologies. *J. Data Semantics*, 10:62–95, 2008.
10. Sik Chun Joey Lam. *Methods for Resolving Inconsistencies In Ontologies*. PhD thesis, Department of Computer Science, Aberdeen, 2007.
11. Marvin Minsky. A framework for representing knowledge. In *The Psychology of Computer Vision*, 1975.
12. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language semantics and abstract syntax. W3C Recommendation, 10 February 2004.
13. Stefan Schlobach and Ronald Cornet. Explanation of terminological reasoning a preliminary report. In Diego Calvanese, Giuseppe De Giacomo, and Enrico Franconi, editors, *Proceedings of the 2003 International Workshop on Description Logics (DL2003), Rome, Italy September 5-7, 2003*, volume 81 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.
14. Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 5(2), 2007.
15. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.