

# Implementing OWL Defaults

Vladimir Kolovski<sup>1</sup> and Bijan Parsia<sup>2</sup> and Yarden Katz<sup>1</sup>

<sup>1</sup> Department of Computer Science,  
University of Maryland, College Park, MD 20742, USA,  
kolovski@cs.umd.edu, yarden@umd.edu

<sup>2</sup> School of Computer Science  
The University of Manchester,  
bparsia@cs.man.ac.uk

**Abstract.** While it has been argued that knowledge representation for the World Wide Web must respect the open world assumption due to the “open” nature of the Web, users of the Web Ontology Language (OWL) have often requested some form of non-monotonic reasoning. In this paper, we present preliminary optimizations and an implementation of a restricted version of Reiter’s default logic as an extension to the description logic fragment of OWL, OWL DL. We implement the decision procedure for this logic in the OWL DL reasoner Pellet, and exploit its support for incremental reasoning through update to improve performance. We also extend an open source ontology editor SWOOP with default rules editing support.

## 1 Introduction

The WebOnt Working Group identified two objectives<sup>3</sup> concerning, at least potentially, non-monotonicity, and yet the resulting Web Ontology Language (OWL) is thoroughly monotonic. Both objectives (some form of defaults, i.e., “default property values”, and an unspecified version of the closed world assumption) could be plausibly met with Reiter’s default logic, which is one of the most studied non-monotonic logics.

Reiter’s default logic (or simply default logic henceforth), while very expressive, is, like many non-monotonic formalisms, known to be computationally difficult even in the propositional case. In [1], Baader and Hollunder showed that even a restricted form of defaults coupled with a description logic that contains a smaller set of class constructors than OWL-DL<sup>4</sup> is undecidable. They also showed that if one restricted the defaults to apply to only to named individuals (or, equivalently, restricted the logic to closed defaults), then a robust decidability ensued. The decision procedure proposed in that paper proved difficult to implement with reasonable efficiency and the proposal has languished ever since.

However, the state of the art in description logic reasoners has advanced, and there have been recent developments in the OWL-DL reasoner Pellet that suggest that a

---

<sup>3</sup> See O2 and O3 in the use cases and requirements document at <http://www.w3.org/TR/webont-req/>

<sup>4</sup> The logic that Baader and Hollunder use ( $\mathcal{ALCF}$ ) incorporates feature agreements, which are not part of OWL-DL, so we cannot claim it is a subset.

reasonable implementation of a default logic reasoner can be achieved. These developments include tableau tracing for the description logic *SHOIN* and incremental reasoning support. In this paper, we present a prototype implementation of the “terminological defaults” of Baader and Hollunder based on the above developments. Tableau tracing allows us to use *SHOIN* as the underlying description logic of the terminological defaults, hence the term OWL defaults. We provide UI for this defaults reasoner by extending open source ontology editor Swoop [10]. We show through empirical evaluation that this prototype performs reasonably overall, but also does fairly well in some cases when compared to the propositional default logic reasoner, XRay.

## 2 Background

### 2.1 Default Logic

Reiter’s default logic is a nonmonotonic formalism for expressing commonsense rules of reasoning. These rules, called default rules (or simply *defaults*), are of the form:

$$\frac{\alpha : \beta}{\gamma}$$

where  $\alpha, \beta, \gamma$  are first-order formulae. We say  $\alpha$  is the *prerequisite* of the rule,  $\beta$  is the *justification* and  $\gamma$  the *consequent*. Intuitively, a default rule can be read as: if I can prove the prerequisite from what I believe, and the justification is consistent with what I believe, then add the consequent to my set of beliefs. For a set of default rules  $\mathcal{D}$ , we denote the sets of formulae occurring as prerequisites, justifications, and consequents in  $\mathcal{D}$  by  $Pre(\mathcal{D})$ ,  $Jus(\mathcal{D})$ , and  $Con(\mathcal{D})$ , respectively.

**Definition 1** *A default theory is a pair  $\langle \mathcal{W}, \mathcal{D} \rangle$  where  $\mathcal{W}$  is a set of closed first-order formulae (containing the initial world description) and  $\mathcal{D}$  is a set of default rules. A default theory is closed if there are no free variables in its default rules.*

Possible sets of conclusions from a default theory are defined in terms of *extensions* of the theory. Extensions are deductively closed sets of formulae that also include the original set of facts from the world description. Extensions are also closed under the application of defaults in  $\mathcal{D}$  - we keep applying default rules as long as possible to generate an extension.

Default rules can conflict. A simple example is when two defaults  $d_1$  and  $d_2$  are applicable yet the consequent of  $d_1$  is inconsistent with the consequent of  $d_2$ . We then typically end up with two extensions: one where the consequent of  $d_1$  holds, and one where the consequent of  $d_2$  holds.

### 2.2 Terminological Default Theories

In this section, we present some definitions from [1] and their algorithm for computing all extensions of a default theory (which was the basis for our implementation).

**Definition 2 (Terminological Default Theory)** *A terminological default theory is a pair  $\langle \mathcal{W}, \mathcal{D} \rangle$  where  $\mathcal{W}$  is a SHOIN knowledge base and  $\mathcal{D}$  is a finite set of default rules whose prerequisites, justifications, and consequents are concept expressions.*

A terminological default theory is not necessarily decidable - in fact, [1] shows that a default theory  $\langle \mathcal{W}, \mathcal{D} \rangle$  where  $\mathcal{W}$  is an  $\mathcal{ALCF}$  logic is undecidable due to skolemization in Reiter's treatment of open defaults. In order to retain decidability, [1] proposes a restricted semantics: default rules are only applicable to named individuals. Thus, a set of defaults  $\mathcal{D}$  is interpreted as representing the closed defaults obtained by instantiating the free variable by all individual names occurring in  $\mathcal{W}$ . Baader and Hollunder also show that as long as the underlying logic language is decidable, with this restricted semantics the problem of computing all of the extensions is also decidable.

**Reasoning with Terminological Defaults** For sake of completeness, we describe in this section the algorithm from [1] for computing all of the extensions of a terminological default theory. Our implementation and optimizations are based on this algorithm.

**Definition 3 (Grounded Set of Defaults)** *Let  $\mathcal{W}$  be a set of closed formulae, and  $\mathcal{D}$  be a set of closed defaults. We define  $\mathcal{D}_0 = \emptyset$  and, for  $i \geq 0$ ,*

$$\mathcal{D}_{i+1} = \mathcal{D}_i \cup \left\{ d = \frac{\alpha : \beta}{\gamma} \mid d \in \mathcal{D} \text{ and } \mathcal{W} \cup \text{Con}(\mathcal{D}_i) \models \alpha \right\}$$

*Then  $\mathcal{D}$  is called grounded in  $\mathcal{W}$  iff  $\mathcal{D} = \bigcup_{i \geq 0} \mathcal{D}_i$*

The main reasoning service provided is the procedure that computes all of the extensions of a terminological default theory  $\mathcal{D}$ . The procedure works in a top-down fashion: it starts with the set of all ground defaults  $\mathcal{D}_0$ , and then inspects if there is a conflicting default  $d$  in that set (i.e., the justification of  $d$  conflicts with  $\mathcal{W} \cup \text{Con}(\mathcal{D}_0)$ ). There are two ways to eliminate the conflict and the procedure does both. First, it simply removes the default  $d$  from  $\mathcal{D}_0$  and applies itself again to the smaller set  $\mathcal{D}_0 \setminus d$ . The other way is to keep  $d$  in  $\mathcal{D}_0$ , and prune other defaults that cause the conflict. In other words, for a conflicting default  $d$ , the procedure is also invoked with the maximal subsets  $\mathcal{D}'$  of  $\mathcal{D}_0$  s.t.  $d \in \mathcal{D}'$  and the justification of  $d$  does not conflict with  $\mathcal{W} \cup \text{Con}(\mathcal{D}')$ . It is shown in [1] how finding the maximal subsets of  $\mathcal{D}_0$  above reduces to the problem of computing maximally consistent subsets: given two sets  $\mathcal{W}$  and  $\mathcal{D}_0$ , find the maximal subsets of  $\mathcal{D}' \in \mathcal{D}_0$  s.t.  $\mathcal{W} \cup \mathcal{D}'$  is consistent.

### 3 Optimizations

To the best of our knowledge there has been no adaptation of Baader and Hollunder's algorithm to OWL-DL. Recently, however, there has been work in Description Logics that has allowed us to adapt the algorithm and optimize it to produce a working implementation. First, there is a practical tableau tracing algorithm [11] with low overhead which can be used solve the problem of computing maximally consistent subsets for

more expressive logics (up to *SHOIN*). Second, there has been recent work on optimizing reasoning for dynamic ABoxes [8]. This work allows us to optimize the consistency check every time there is an update to the working set of defaults  $\mathcal{D}_0$ . Third, we propose a technique for generating possible (*candidate*) extensions in a bottom-up fashion. We discuss all of these techniques in this section.

### 3.1 Finding Maximally Consistent Subsets

In this section, we explain how we apply recent work in tableau tracing and debugging OWL ontologies to solve the problem of finding the maximal subsets  $\mathcal{D}' \in \mathcal{D}$  such that  $Con(\mathcal{D}') \cup \mathcal{W}$  is consistent. The approach in [1] introduces clash formulae for this problem - instead we use *justifications*. A justification for an ABox inconsistency is a minimal fragment of the KB in which the ABox is inconsistent. Justifications are computed by a tracing function that records the changes in the tableau and the axioms and non-deterministic choices that are responsible for those changes. We refer the reader to [11] for more details.

Every justification contains DL axioms which are responsible for the particular clash. In our case, these axioms come either from  $\mathcal{W}$  or  $Con(\mathcal{D})$ . A justification where all of the axioms come from  $\mathcal{W}$  indicates that  $\mathcal{W}$  by itself is inconsistent. In this paper, without loss of generality, we will assume that  $\mathcal{W}$  is consistent<sup>5</sup>. Thus, every justification will contain at least one axiom from  $Con(\mathcal{D})$ .

In order to find the maximally consistent subsets of  $Con(\mathcal{D})$ , the first step is to find all of the justifications for the inconsistency of  $\mathcal{W} \cup Con(\mathcal{D})$ . Using the information from those justifications, then we can determine the minimum number of axioms coming from  $Con(\mathcal{D})$  that should be removed to render  $\mathcal{W} \cup Con(\mathcal{D})$  consistent. We use the algorithm from [9] to find all justifications: it does tableau tracing to find one justification and Reiter's Hitting Set Tree (HST) algorithm [14] to find all other justifications. The intuition behind the usage of the Hitting Set relies on the fact that in order to make the KB consistent, one needs to remove from the KB at least one axiom from each justification.

After computing all justifications, the next step is to determine what is the smallest set of axioms that we need to remove to make it consistent. Because of the nature of the justifications, we only need to remove one axiom from each justification to make the KB consistent. It is important to note here that even though the axioms in a justification can come from either  $\mathcal{W}$  or  $Con(\mathcal{D})$ , we are only interested in removing axioms from  $Con(\mathcal{D})$ . Thus, we pre-process each justification, discarding the axioms coming from  $\mathcal{W}$  and only leaving the axioms from  $Con(\mathcal{D})$ .

At this point, we apply Reiter's HST algorithm again on the pruned justifications to obtain the minimum set cover. For example, for the set of justifications  $S = \{\{a\}, \{a, b\}, \{c\}, \{c, d\}\}$  the minimal hitting set is  $\{a, c\}$ . This minimal hitting set gives us the smallest set of axioms from  $Con(\mathcal{D})$  that we need to remove to make  $Con(\mathcal{D}) \cup \mathcal{W}$  consistent. Note that at this point, there are no consistency checks performed while computing the minimal hitting sets (unlike the previous application of Reiter's algorithm) and the justifications

<sup>5</sup> In the case  $\mathcal{W}$  is inconsistent, then there is only one extension consisting of all of the formulae

contain less. Nonetheless, we have observed that this task is the bottleneck of the system due to the exponential nature of the hitting set tree algorithm.

### 3.2 ABox Updates

In the procedure described in Section 2.2 often there are incremental changes to the working set of defaults  $\mathcal{D}_0$ : either the conflicting defaults is removed, or a minimal set of other defaults that cause the conflict. These modifications are equivalent of removing (and adding) type assertions to  $\mathcal{W} \cup \text{Con}(\mathcal{D}_0)$ , thus reasoning time overall would improve if we used a DL reasoner optimized for ABox updates.

**Incremental Consistency Checking** The recent work that we used was [8], which is an approach for incrementally updating tableau completion graphs under ABox updates. Their idea is not to throw away the old completion graph after an update. Instead, the update algorithm adds new (resp. removes existing for deletions) components (edge and/or nodes) induced by the update to the old completion graph; after this, standard tableau completion rules are re-fired to ensure that the model is complete. Therefore the previous completion is updated such that if a model exists (i.e, the KB is consistent after the update) a new completion graph will be found. In [8], it was observed that updates did not have a large effect on the existing completion graph, therefore orders of magnitude performance improvements are achieved.

**Applying Incremental Reasoning** The incremental reasoning techniques can be applied in a number of places in the algorithm from [1] to improve overall performance. First, we use them to compute the largest ground subset of rules - the pseudo-code for this procedure is given in Figure 3.2.

```

function Find-Largest-Grounded-Subset( $\mathcal{W}, \mathcal{D}$ )
begin
(1)  $\mathcal{D}_l = \emptyset, \text{changed} = \text{true}$ 
(2) while  $\text{changed}$  do
(3)    $\text{changed} = \text{false}$ 
(4)   foreach  $d_i = \alpha : \beta/\gamma \in \mathcal{D} \setminus \mathcal{D}_l$ 
(5)     if  $\mathcal{W} \cup \text{Con}(\mathcal{D}_l) \models \alpha$  then
(6)        $\text{incupdate}(\mathcal{W}, \gamma, \text{add})$ 
(7)        $\mathcal{D}_l = \mathcal{D}_l \cup \{d_i\}$ 
(7)        $\text{changed} = \text{true}$ 
(8) return  $\mathcal{D}_l$ 
end

```

**Fig. 1.** Pseudo-code to get the largest grounded subset. It is important to note line (6) which incrementally updates the world description  $\mathcal{W}$  with a new type assertion. In line (5), to check whether  $\mathcal{W} \cup \text{Con}(\mathcal{D}_l) \models \alpha$  we simply  $\text{incupdate}(\mathcal{W}, \neg\alpha, \text{add})$  and inspect the updated completion graph.

Another application of the ABox updates is in the procedure of finding all justifications (Section 3.1). While building the hitting set tree, instead of rebuilding the completion graph from scratch (when removing or adding axioms) we update only those portions affected by the change.

### 3.3 Generating Candidate Extensions

Considering that the bottleneck of the algorithm that computes the extensions is in the subtask that computes the maximally consistent subsets (empirical results are in Section 5), we introduce here another optimization aimed at reducing the number of calls to the afore-mentioned subtask. The purpose of this optimization is to find the obvious conflicts and generate approximate extensions before the algorithm in [1] is used. The optimization is based on the idea of generating extensions bottom-up, as explored in [12, 3]. Defaults that conflict are placed into separate extensions. However, when determining whether defaults  $d_1$  and  $d_2$  conflict, we simplify the conflict detection by taking only the world description  $\mathcal{W}$  into account, and ignoring the current extension. Following we provide a definition of obviously conflicting defaults.

**Definition 4 (Obviously Conflicting Rules)** *For a default theory  $\langle \mathcal{W}, \mathcal{D} \rangle$  we define two closed defaults  $d_1, d_2 \in \mathcal{D}$  to be obviously conflicting iff at least one of the following holds:*

$$\begin{aligned} \mathcal{W} \cup \text{Con}(d_1) \models \neg \text{Jus}(d_2), \quad \mathcal{W} \cup \text{Con}(d_1) \models \neg \text{Con}(d_2), \\ \mathcal{W} \cup \text{Con}(d_2) \models \neg \text{Jus}(d_1) \text{ or } \mathcal{W} \cup \text{Con}(d_2) \models \neg \text{Con}(d_1) \end{aligned}$$

To find all of the obvious conflicts, we use a naive  $O(N^2)$  algorithm where  $N$  is the number of closed defaults. Note that in this case we can again exploit incremental reasoning techniques. Using the information about the conflicts, we proceed to generate possible extensions. Since the extensions are built only on the basis of the simplified idea of obvious conflicts, and are different from extensions in terminological default theories, we call them *candidate* extensions.

**Definition 5 (Candidate Extension)** *Let  $T = \langle \mathcal{W}, \mathcal{D} \rangle$  be a closed default theory. Let  $\mathcal{D}_g = \{\alpha : \beta/\gamma \mid \alpha : \beta/\gamma \in \mathcal{D} \text{ and } \mathcal{W} \models \alpha\}$ .*

*Let  $\mathcal{S}_0 = \emptyset$  and  $\forall i \geq 0$  define:*

$$\mathcal{S}_{i+1} = \mathcal{S}_i \cup \{d \mid d \in \mathcal{D}_g \text{ and there is no } h \text{ s.t. } h \in \mathcal{S}_i \text{ and } \text{conflict}(d, h)\}$$

*Then  $\mathcal{S} \cup \{\mathcal{D} \setminus \mathcal{D}_g\}$  is a set of generating defaults of a candidate extension iff  $\mathcal{S} = \bigcup_{i \geq 0} \mathcal{S}_i$ .*

To compute the candidate extensions, we follow the above definition: working only with the set of defaults that is ground w.r.t. the world description  $\mathcal{W}$ , we build candidate extensions using backtracking search making sure that no two "obviously" conflicting defaults are in the same extension. Then to each generated extension we append the set of non-ground defaults (defaults not used in the search). These candidate extensions are passed to the main (top-down) procedure where the rest of the conflicts are found and pruned. We have evaluated the impact of candidate extensions (results available in Section 5) and the initial results are encouraging.

## 4 Implementation

We have a preliminary implementation of our optimizations. It is provided as an extension to an open source DL reasoner [13] and it provides *realization* of individuals with terminological default rules, with either credulous or skeptical reasoning. We have also provided a UI for defaults by extending the open source OWL Ontology editor Swoop [10]. More specifically, we added support for default rules editing (shown in Figure 2) and informing the user of the inferences made by the rules. Also, we update the current ontology with the set of inferred facts from the defaults.

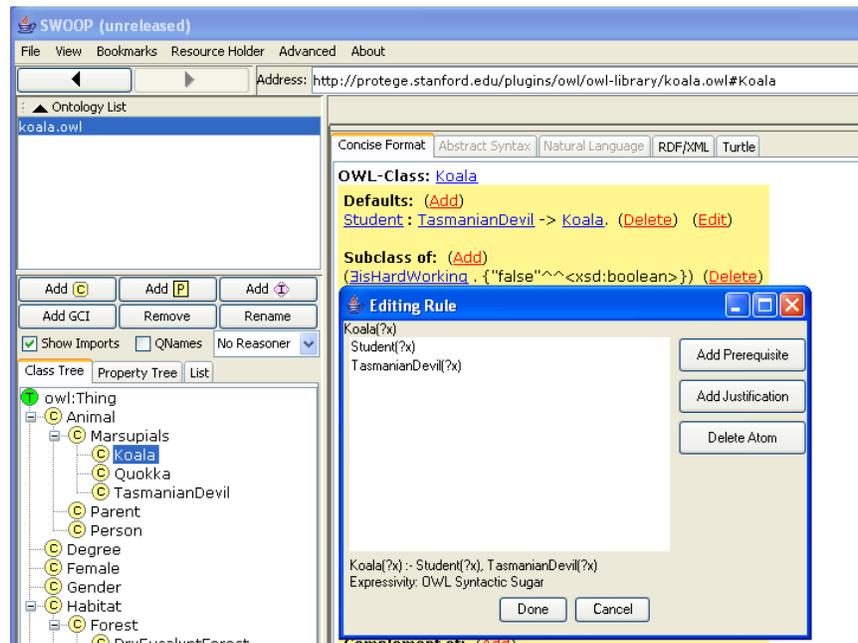


Fig. 2. Screenshot of defaults editor

## 5 Evaluation

To evaluate our implementation, we used an IBM ThinkPad T42 with Pentium Centrino 1.6GHz processor and 1.5GB of memory. First we show the performance benefits of incremental reasoning support and generating candidate extensions. The ontologies used for these tests were modified versions of the Lehigh University Benchmark (LUBM)[7], with two open default rules of the Nixon diamond shape. The number of extensions is  $2^i$  (Column *ext.*), where  $i$  is the number of 'Nixon' individuals. Performance results (times in seconds) are shown in Table 1. *Cand\_ext* is the time spent generating the candidate

extensions, *grounded* is the time spent computing largest grounded subsets, and *HST* is the time spent computing maximally consistent subsets. The benefit of candidate extensions becomes obvious in cases with more extensions for a default theory.

Size		fully-optimized			updates-only			not-optimized		
inds	ext.	cand_ext	HST	total	grounded	HST	total	grounded	HST	total
50	8	.291	.300	.631	.803	.770	1.513	.952	1.671	2.553
100	8	.731	.311	1.122	1.192	.861	2.003	2.585	1.902	4.447
200	8	2.504	.430	3.084	1.983	1.101	3.165	11.446	2.934	14.190
50	64	.291	1.241	1.933	3.547	4.364	8.462	3.466	10.365	14.141
100	64	.761	1.452	2.763	4.408	5.086	10.515	5.987	14.052	21.601
200	64	2.613	1.993	<b>5.597</b>	6.611	7.610	<b>16.614</b>	20.292	20.476	<b>43.072</b>

**Table 1.** Comparison of three versions of pellet-defaults. **Not-optimized** does not generate any candidate extensions or perform reasoning with updates. **Updates-only**, includes the incremental reasoning support but not the candidate extensions. **Fully-optimized** includes all of the techniques presented in this paper.

Next, we present a comparison of our tool with XRay [5], which is a popular Prolog-based implementation for query-answering in *propositional* default logics, and smodels [12], an implementation of the stable model semantics for logic programs with negation as failure. The first set of defaults rules that we used was a set of sample defaults on XRay’s homepage [6]:

$$d_1 = \frac{Student : Student}{Adult} \quad d_2 = \frac{Student : Student}{\neg Employed} \quad d_3 = \frac{Student : Adult}{\neg Married}$$

$$d_4 = \frac{Adult : \neg Student}{Employed} \quad d_5 = \frac{Adult : \neg Student}{Married}$$

This theory has only one extension (where only  $\{d_1, d_2, d_3\}$  are fired for each named individual). Unfortunately, since (to the best of our knowledge) smodels does not support classical negation, we could not evaluate its performance for these test cases.

The second test case is a default theory with only two open defaults, of the Nixon diamond shape. Results are shown in Table 2. In the case when there are few extensions, our approach outperforms XRay, even though our algorithm is not specifically tuned for propositional default logics. In the second example, in cases with a lot of extensions, we do not fare as well.

## 6 Related Work

The majority of reasoners for default logic focus on *propositional* default theories, where quantifiers are not allowed. Two capable systems in this group that are com-

Type and Size of KB			Pellet-defaults			XRay	smodels
Type	Size	Ext.	grounded	HST	total	total	total
xray ex.	50	1	.181	.370	.615	1.472	N/A
	100	1	.5	.541	1.090	4.637	N/A
	500	1	15.142	1.712	15.252	timeout	N/A
Nixon	3	8	.03	.15	.2	.03	.02
	6	64	.130	.280	.63	.11	.04
	11	2048	2.183	3.375	1.0745	.180	1.444

**Table 2.** Evaluation Results for computing all of the extensions.

parable in functionality include XRay [5] and DeReS [3]. However, the restricted expressivity of the languages these reasoners handle make them of limited interest for OWL reasoning. It would be unrealistic to ignore many of OWL’s key constructs—e.g. existential, universal, and cardinality restrictions—for the benefit of adding default rules.

Smodels [12] and dl<sub>v</sub> [4] are both efficient implementations of the answer set semantics. Since the answer set (or stable models) semantics for logic programs can be embedded into Reiter’s default logic (an answer set is analogous to a default theory extension), these systems can also be seen as default logic reasoners. The main difference with our approach is that we are description logic-based, whereas they use logic programming techniques.

## 7 Discussion and Future Work

It is obvious from the evaluation that our approach does not work well with default theories with lots of extensions. The reason is that the number of extensions is determined by the number of conflicting rules. Every time a conflicting default is detected, the subtask to compute the maximally consistent subsets procedure is executed (which is the bottleneck of our system). A goal for future optimizations could be to avoid calling this subtask as much as possible. We have taken the first step with the bottom-up generation of candidate extensions. However, further work is needed. We believe it is possible to achieve much more accurate and efficient generation of candidate extensions, based on the techniques in [12, 3, 2].

In terms of expressivity, an important next step would be to handle one of the many *prioritized default logics*, where default rules are partially ordered to determine preferred extensions.

## 8 Acknowledgements

Thanks to Christian Halaschek-Wiener, Aditya Kalyanpur and Evren Sirin for helpful comments and discussions. This work was supported in part by grants from Fujitsu,

Lockheed Martin, NTT Corp., Kevric Corp., SAIC, the National Science Foundation, the National Geospatial-Intelligence Agency, DARPA, US Army Research Laboratory, and NIST.

## References

1. Franz Baader and Bernhard Hollunder. Embedding Defaults Into Terminological Knowledge Representation Formalisms. *J. Autom. Reasoning*, 14(1):149–180, 1995.
2. Pawel Cholewinski. Reasoning with stratified default theories. In *LPNMR '95: Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 273–286, London, UK, 1995. Springer-Verlag.
3. Pawel Cholewinski, V. Wiktor Marek, and Mirosław Truszczyński. Default reasoning system *deres*. In *Principles of Knowledge Representation and Reasoning*, pages 518–528, 1996.
4. Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR system *d1v*: Progress report, comparisons and benchmarks. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann, San Francisco, California, 1998.
5. T. Schaub et al. XRay: A prolog technology theorem prover for default reasoning: A system description. In *Proc. of the Thirteenth International Conference on Automated Deduction (CADE)*, pages 293–297, 1996.
6. Torsten Schaub et al. XRay: User's guide & reference manual. <http://www.cs.uni-potsdam.de/wv/xray/>.
7. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.
8. Christian Halaschek-Wiener, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Description logic reasoning for dynamic aboxes. In *Proc. of the Int. Description Logic Workshop (DL)*, 2006.
9. Aditya Kalyanpur. Debugging and repair of owl ontologies. In *Ph.D. Dissertation, University of Maryland, College Park*. <http://www.mindswap.org/papers/2006/AdityaThesis-DebuggingOWL.pdf>.
10. Aditya Kalyanpur, Bijan Parsia, and James Hendler. A tool for working with web ontologies. In *In Proceedings of the International Journal on Semantic Web and Information Systems, Vol. 1, No. 1, Jan - March*, 2005.
11. Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. In *Journal of Web Semantics - Special Issue of the Semantic Web Track of WWW2005*, 2005. (To Appear).
12. Ilkka Niemelä; and Patrik Simons. Smodels - an implementation of the stable model and well-founded semantics for normal lp. In *LPNMR '97: Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 421–430, London, UK, 1997. Springer-Verlag.
13. Bijan Parsia and Evren Sirin. Pellet: An OWL DL reasoner. In *Third International Semantic Web Conference - Poster*, 2004.
14. Ray Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.