On the Semantics of Trust and Caching in the Semantic Web

Simon Schenk

ISWeb, University of Koblenz-Landau, Germany
sschenk@uni-koblenz.de
http://isweb.uni-koblenz.de/

Abstract. The Semantic Web is a distributed environment for knowledge representation and reasoning. The distributed nature brings with it failing data sources and inconsistencies between autonomous knowledge bases. To reduce problems resulting from unavailable sources and to improve performance, caching can be used. Caches, however, raise new problems of imprecise or outdated information. We propose to distinguish between certain and cached information when reasoning on the semantic web, by extending the well known $\mathcal{FOUR}$ bilattice of truth and knowledge orders to $\mathcal{FOUR}-\mathcal{C}$, taking into account cached information. We discuss how users can be offered additional information about the *reliability* of inferred information, based on the availability of the corresponding information sources. We then extend the framework towards $\mathcal{FOUR}-\mathcal{T}$, allowing for multiple *levels of trust* on data sources. In this extended setting, knowledge about trust in information sources can be used to compute, how well an inferred statement can be trusted and to resolve inconsistencies arising from connecting multiple data sources. We redefine the stable model and well founded semantics on the basis of $\mathcal{FOUR}-\mathcal{T}$, and reformalize the Web Ontology Language OWL2 based on logical bilattices, to augment OWL knowledge bases with trust based reasoning.

The Semantic Web is envisioned to be a *Web of Data* [2]. As such, it integrates information from various sources, may it be through rules, data replication or similar mechanisms. Obviously, in a distributed scenario, information sources may become unavailable. In order to still be able to answer queries in such cases, mechanisms like caching can be used to reduce the negative implications of failure. Alternatively, some default truth value could be assumed for unavailable information. However, cached values may be inaccurate or outdated, default assumptions can be wrong. Moreover, also available information sources may be trusted to different extents. We propose a framework for reasoning with such trust levels, which allows to give additional information on the reliability of results to the user. In particular, we are able to tell whether a statement's truth value is inferred based on really accessible information, or whether it might change in the future, when cached or default values are updated. Consequently, we extend the framework towards multiple levels of trust, taking into account a *trust order* over information sources, which can possibly be *partial*. While

assigning absolute trust values has little semantics, users are usually good at comparing the trustworthiness of *two* information sources.

This problem is highly relevant, because fault tolerance and reliable data integration is a main prerequisite for a distributed system like the semantic web. The level of reliability of a piece of information can strongly influence further usability of derived information. For this reason, our approach can be seen as a bridge between the rules, proof and trust layers of the semantic web layercake.

The availability of multiple integration mechanisms for distributed resources makes formulating a generic framework a non-trivial task. Moreover, classical two-valued logic fails to capture the 'unknown' truth value of unavailable information. In fact, many applications today rely on simple replication of all necessary data, instead of more flexible mechanisms.

Our approach extends a very flexible basis of most logical frameworks, namely bilattices, which allow to formalize many logics in a coherent way [9]. Hence, it is applicable to a broad range of logical languages. We propose extensions $\mathcal{FOUR}-\mathcal{C}$ and $\mathcal{FOUR}-\mathcal{T}$ to the the $\mathcal{FOUR}$ bilattice. These extensions add trust orders to truth values. As we allow possibly partial orders and also use multiple $\top$ and $\bot$ values, $\mathcal{FOUR}-\mathcal{T}$ is strictly more expressive than for example fuzzy logics.

We investigate support for connected and interlinked autonomous and distributed semantic repositories (for RDF or OWL) — a basic idea behind the semantic web effort. These repositories exchange RDF and OWL data statically (e.g. by copying whole RDF graphs, as in the caching scenario) or dynamically using views or rules. We model our example scenario as follows: Information sources consist of facts (à la RDF) or axioms (à la OWL). Information sources are connected through views, which are clauses of normal programs. This for example subsumes SPARQL queries and SPARQL based views [13], [16]. One possible syntactic and semantic realization of this scenario are Networked Graphs, which we describe in [16].

We show how $\mathcal{FOUR}-\mathcal{C}$ and $\mathcal{FOUR}-\mathcal{T}$ can be used for an extension of the stable model and well founded semantics, which are very popular for rule based mechanisms and also underlie our Networked Graphs mechanism. We extend the web ontology language OWL2 [8] to be based on logical bilattices and use trust levels for inconsistency resolution in OWL2. Finally, we review related work in section 8 before concluding the paper.

1 Use Case

Oscar is a project officer at a large funding agency and supervises several research projects. One of his regular tasks is to check the timely publication of project deliverables. Fortunately, the Semantic Web has made his life much easier. All his projects are advised to publish their deliverables on their websites using the FOAF vocabulary[1].

[1] http://xmlns.com/foaf/spec

In the following, we annotate predicates $p(x)$ with the information *source*, where the corresponding data is expected to come from as follows: $p(x)|source$. Rules and DL axioms A are written in the usual syntaxes. Analogously, we annotate them with the repositories they are stored in as follows: $A||source$. We use functional style syntax for all facts.

Oscar sets up a view listing all deliverables of all projects and their due dates. The view makes use of his own data and the projects' websites to determine all timely deliverables.

(1) $deliverable(report1).||oscar$

(2) $due(report1, 20081005).||oscar$

(3) $hasDeliverable(project1, report1).||oscar$

(4) $TimelyDeliverable(X) \leftarrow deliverable(X)|oscar, due(X, Y)|oscar,$
$published(X, Z)|project1 \vee project2, Z \leq Y|oscar.||oscar$

Rule (4) uses information from Oscars knowledge base to find all deliverables and due dates, as well as information about publishing dates from the project websites, to infer whether deliverables are published on time. When it is time for Oscar's next check, he opens his knowledge base and lists all delayed deliverables. Unfortunately on this day, Project1's website is not working due to technical problems. However, Oscar's webserver still has partial results cached from last month. Now Oscar would like to be able to infer, which deliverables have been delivered, and which information about deliveries might be outdated. He does not want to send a formal reminder to Project1 by mistake. The next day, Project1's website works again, listing

(5) $published(report1, 20081001).||project1$

Oscar wants to easily produce reports, so he adds the following DL axioms to distinguish between good and bad projects, and some additional facts:

(5) $GoodProject = Project \sqcap \forall hasDeliverable.TimelyDeliverable.||oscar$

(6) $BadProject = Project \sqcap \neg GoodProject.||oscar$

(7) $hasDeliverable(project1, report2).||oscar$

(8) $\leq 2\, project1.hasDeliverable.||oscar$

(9) $report1 \not\approx report2.||oscar$

(10) $timelyDeliverable(report2).||oscar$

(11) $Project(project1).||oscar$

When Oscar goes on vacation, he asks his secretary Susan to monitor deliverables, while he is away. Susan does her own bookkeeping, also using OWL. When Oscar returns, he imports Susan's data into his knowledge base, causing an inconsistency, as Susan has added an axiom

(12) $\neg timelyDeliverable(report2).||susan$

As Oscar assumes his own data to be more reliable, he would like to automatically resolve such inconsistencies in the future by discarding lowly trusted information. Additionally, he wants to make sure, that projects can not cheat. He trusts projects less than his secretary. As he does not prefer any project over others, however, there is no trust order among projects. So if we have

(13) $SuccessfulProject(project1).||project1$ and
$\neg SuccessfulProject(project1).||project2$,

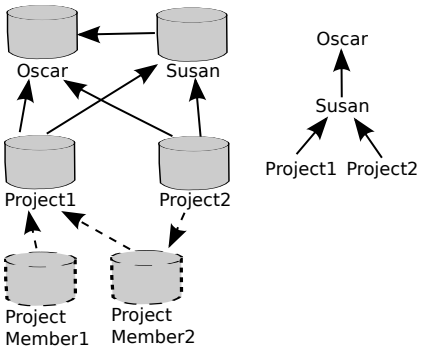

Fig. 1. Repositories in Use Case (left) and Oscar's Trust Order (right)

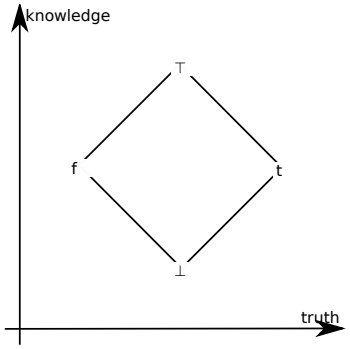

Fig. 2. The knowledge and truth order $\mathcal{FOUR}$.

he wants to discard both axioms, while in the case of

(14) $SuccessfulProject(project1).||project1$ and $\neg SuccessfulProject(project1).||susan$,

only $SuccessfulProject(project1).||project1$ would be discarded.

All repositories involved in this scenario are shown in the left part of fig 1. In the lower part we see that also additional repositories might be involved, which are not directly relevant and known to Oscar. In the right part, we see Oscar's trust order. Obviously, there is a need for multiple levels of trust in information sources in our scenario. However, not all sources need to be comparable. In this paper, we propose a very flexible mechanism for reasoning with such partial trust orders.

2 $\mathcal{FOUR}$

Most logic programming paradigms, including classical logic programming, stable model and well founded semantics, and fuzzy logics can be formalized based on bilattices of truth values and fixpoints of a direct consequence operator on such a bilattice. Therefore, if we build our extension into this foundational layer, it will directly be available in many different formalisms.

A logical bilattice [5] is a set of truth values, on which two partial orders are defined, which we call the truth order $\leq_t$ and the knowledge order $\leq_k$. Both $\leq_t$ and $\leq_k$ are complete lattices, i.e. they have a maximal and a minimal element and every two elements have exactly one supremum and infimum.

In logical bilattices, the operators $\vee$ and $\wedge$ are defined as supremum and infimum wrt. $\leq_t$. Analogously join ($\oplus$) and meet ($\otimes$) are defined as supremum and infimum wrt. $\leq_k$. As a result, we have multiple distributive and commutative laws, which all hold. Negation ($\neg$) simply is an inversion of the truth order. Hence, we can also define material implication ($a \rightarrow b = \neg a \vee b$) as usual.

The smallest non trivial logical bilattice is $\mathcal{FOUR}$, shown in figure 2. In addition to the truth values t and f, $\mathcal{FOUR}$ includes $\top$ and $\bot$. $\bot$ means "unknown", i.e. a fact is neither true or false. $\top$ means "overspecified" or "inconsistent", i.e. a fact is both true and false.

In traditional, two valued logic programming without negation, only t and f would be allowed as truth values. In contrast, e.g. the stable model semantics, allows to use $\top$ and $\bot$. In this case, multiple stable models are possible. For example, we might have a program with three clauses:

$man(bob) \leftarrow person(bob), \neg woman(bob)$.

$woman(bob) \leftarrow person(bob), \neg man(bob)$.

$person(bob)$.

Using default f, we might infer both $man(bob) \wedge \neg woman(bob)$ and $woman(bob) \wedge \neg man(bob)$. While in two valued logics we would not be able find a model, in four values, we could assign truth values $t \oplus f = \top$ and $t \otimes f = \bot$. In fact, both would be allowed under the stable model semantics, resulting in multiple models for a single program.

The well founded semantics distinguishes one of these models — the minimal one, which is guaranteed to always exist and only uses t, f, and $\bot$. In a similar way, other formalisms can be expressed in this framework as well. Particularly, we can also formalize open world based reasoning, using $\bot$ instead of f as default value. In order to keep this paper short, we refer the reader to the very good overview in [3].

3 $\mathcal{FOUR} - \mathcal{C}$

To apply our work to a variety of different logical formalisms, we directly extend $\mathcal{FOUR}$ as the theoretical basis. We will give two examples of how the extensions can be used by applying them to the well founded semantics and to OWL in the remainder of this paper.

To distinguish between certain information, which is local or currently available online, and cached information (or information derived from cached information), we extend the set of possible truth values: For information, of which we know the actual truth value we use the truth values $\{t_k, f_k, \top_k, \bot_k\}$. For cached information, we use a different set: $\{t_c, f_c, \top_c, \bot_c\}$. The basic idea of the extension is that cached information is always potentially outdated. For example a cached *false* value might actually be *true*. Therefore, we assume cached information to be always a bit less false or true than certain information — as the truth value might have changed.

In our scenario, let us assume Project1's website is currently inaccessible. In a normal closed world setting, we would assume $published(report1, \_)$ to be f, hence also $timelyDeliverable(report1)$, by rule (4). Changing our default to $\bot$ – unknown – would also not help us determine, whether Project1's website is just updated slowly, or whether the available information might be inaccurate. In $\mathcal{FOUR} - \mathcal{C}$, we assign f_c (or $\bot_c$ in an open world setting) to $published(report1, \_)$. We can then conclude from (1-4) and the unavailability of (5) that $timelyDeliverable(report1)$ is $t_k \wedge f_c = f_c$, and hence possibly outdated. Therefore, Oscar will simply update his report later, when all relevant data sources are available again, instead of sending a reminder by mistake. Analogously, if we run into an inconsistency, we want to be aware, if this inconsistency

could potentially be resolved by updating the cache. Summarizing, our operators should act as in $\mathcal{FOUR}$, if we only compare truth values on the same trust level. If we compare values from multiple trust levels, we would like to come up with analogous truth values as in the four valued case, but on the trust level, which is the lowest of the compared values.

Ginsberg [5] describes how we can obtain a logical bilattice: Given two distributive lattices $\mathcal{L}_1$ and $\mathcal{L}_2$, create a bilattice $\mathcal{L}$, where the nodes have values from $\mathcal{L}_1 \times \mathcal{L}_2$, such that the following orders hold:

- $\langle a, b\rangle \leq_k \langle x, y\rangle$ iff $a \leq_{\mathcal{L}_1} x \wedge b \leq_{\mathcal{L}_2} y$ and
- $\langle a, b\rangle \leq_t \langle x, y\rangle$ iff $a \leq_{\mathcal{L}_1} x \wedge y \leq_{\mathcal{L}_2} b$

If $\mathcal{L}_1$ and $\mathcal{L}_2$ are infinitely distributive — that means distributive and commutative laws hold for infinite combinations of the lattice based operators from section 2 — then $\mathcal{L}$ will be as well.

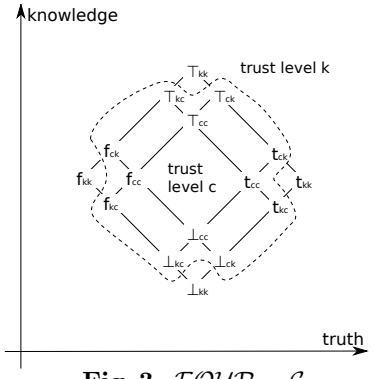

Fig. 3. $\mathcal{FOUR} - \mathcal{C}$

We use $\mathcal{L}_1 = \mathcal{L}_2 = t_k > t_c > f_c > f_k$ as input lattices, resulting from our basic idea that cached values are a bit less true and false. As $\mathcal{L}_1$ and $\mathcal{L}_2$ are totally ordered sets, they are complete lattices and hence infinitely distributive. The resulting $\mathcal{FOUR} - \mathcal{C}$ bilattice shown in fig. 3. In fig. 3 we label nodes of the form $\langle f_x, t_y\rangle$ with $\top_{xy}$, $\langle t_x, f_y\rangle$ with $\bot_{xy}$, $\langle f_x, f_y\rangle$ with f_{xy} and $\langle t_x, t_y\rangle$ with t_{xy}.

The artificial truth values $f_{kc}, f_{ck}, t_{kc}, t_{ck}$, $\top_{kc}, \top_{ck}, \bot_{kc}$ and $\bot_{ck}$ are only used for reasoning purposes. Users will only be interested in trust levels, which are equivalence classes of truth values: Given an order of $k > c$, the trust level of a truth value is the minimal element in its subscript. For example the trust level of $t_c, \bot_{kc}$ and $\top_{ck}$ is c. In fig. 3, these equivalence classes are separated by dotted lines. As we only have two trust levels here, there are exactly two equivalence classes - one for currently accessible (truth values $t_{kk}, f_{kk}, \top_{kk}, \bot_{kk}$) and one for cached information (truth values $t_{cc}, f_{cc}, \top_{cc}, \bot_{cc}$) and information derived from cached information (truth values $t_{kc}, f_{kc}, \top_{kc}, \bot_{kc}, t_{ck}, f_{ck}, \top_{ck}, \bot_{ck}$).

Obviously, $\mathcal{FOUR} - \mathcal{C}$ meets our requirements from the beginning of this section: We have two sub-bilattices isomorphic to $\mathcal{FOUR}$, one on each trustlevel. Additionally, we always come up with truth values on the right trust level, e.g. $f_{kk} \oplus t_{cc} = \top_{kc}$, which is on trust level c and $\top_{kk} \wedge \top_{cc} = \top_{ck}$, which is on trust level c and correctly reflects the fact that the result may be inaccurate in case $\top_{cc}$ needs to be corrected to some f_{xx}.

In the caching scenario, we can assume a default truth value of $\bot$ or f (depending on whether we do open or closed world reasoning) to all statements, where the actual truth value can not be determined at the moment. However, some more information may be available for example due to caching, statistics or similar. Using $\mathcal{FOUR} - \mathcal{C}$, we can still do inferencing in the presence of such unreliable sources. Moreover, a user or application can determine, whether a

piece of information is completely reliable, or if more accurate information may become available.

4 Extension Towards Trust Levels

In the previous chapter we have focused on two levels of reliability of information. More generally, we would like to be able to infer multiple levels of trust in a distributed setting, as we have seen in the use case in Oscar's trust order.

Definition 1 (Trust Order).
A trust order $\mathcal{T}$ is a partial order over a finite set of information sources with a maximal element, called ∞.

∞ is the information source with the highest trust level, assigned to local data. For any two information sources a and b comparable wrt. $\mathcal{T}$, we have a $\mathcal{FOUR}-\mathcal{C}$ lattice as described above, with the less trusted information source corresponding to the inner part of the lattice. Extended to multiple information sources, this results in a situation as depicted in fig. 4, where the outermost bilattice corresponds to local, fully trusted information.

If a and b are not comparable we introduce virtual information sources inf_{ab} and sup_{ab}, such that

- $inf_{ab} < a < sup_{ab}$ and $inf_{ab} < b < sup_{ab}$;
- $\forall_{c<a,c<b} : c < inf_{ab}$ and
- $\forall_{d>a,d>b} : d > sup_{ab}$

To understand the importance of this last step, assume that $c > a > d$ and $c > b > d$ and a, b are incomparable. Then the truth value of $a \vee b$ would have a trust level of c, as c is the supremum in the trust order. Obviously this escaping to a higher trust level is not desirable. Instead, the virtual information sources represent that we need to trust both, a and b, if we believe in the computed truth value. We illustrate this situation in fig. 5 (We abbreviate inf_{ab} by $<$ and sup_{ab} by $>$).

In the general case (≥ 3 incomparable sources) such a trust order results in a non-distributive lattice. This can be fixed, however, by introducing additional virtual nodes. The basic idea here is to create a virtual node for each element in the powerset of the incomparable sources, with set inclusion as the order. We will call this modified trust order *completed*. We can again derive a complete lattice from the completed trust order. As it can become quite large, we only show for $\top$ how a fragment of the logical bilattice is derived from the trust order for the case of two incomparable information sources in fig. 5.

Using the same method as in the previous chapter, we can construct the corresponding bilattice from a given completed truth order as follows: Given a trust order $\mathcal{T}$, generate a lattice $\mathcal{L}_1$, such that

- $f_a <_{\mathcal{L}_1} f_b$, iff $a >_{\mathcal{T}} b$;
- $t_b <_{\mathcal{L}_1} t_a$, iff $a >_{\mathcal{T}} b$ and
- $\forall a : f_a <_{\mathcal{L}_1} t_a$.

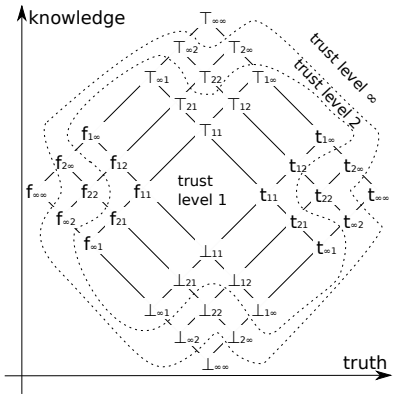

Fig. 4. $\mathcal{FOUR} - \mathcal{T}$

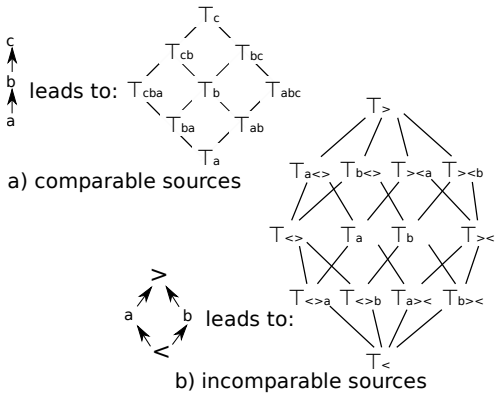

Fig. 5. Virtual Sources

The result is a lattice with t_∞ and f_∞ as maximal and minimal elements. Now create the logical bilattice $\mathcal{L}$ from $\mathcal{L}_1 \times \mathcal{L}_1$ as described in the previous section.

In our scenario, (10) and (12) (as well as (13) and (14)) are inconsistent. We could obtain a consistent knowledge base by removing any of the axioms, however we need to choose which one. Making use of the trust order, we can choose to discard the axiom obtained from Susan. In chapter 6 we define how to do such choices for arbitrarily complex ontologies in an extension of $\mathcal{SROIQ}$. Please note that in spite of the rather simple example, we can have complex derived facts instead and also multiple ways to derive them, as shown in the dependency graph in the left part of in fig. 1.

5 Extended Stable and Well Founded Semantics

The stable model semantics is popular for assigning models to normal logic programs (which are used for modeling our scenario) without limitations on the use of negation. For every normal program there is a distinguished stable model, called the well founded model. We now define a well-founded semantics [19], which takes into account the extensions defined in the previous chapter. We follow the definitions in [3] and [14], which are based on $\mathcal{FOUR}$ and $\{f, \bot, t\}$ respectively. As we use complete, distributive bilattices, the results extend to $\mathcal{FOUR} - \mathcal{T}$ [3], [5]. In the following we refer to normal logic programs when using the term program. In fact, it would even be possible to use $\oplus$ and $\otimes$ as operators in programs.

Definition 2 (Valuations).

Let P be a program. ground(P) *is the set of ground instantiated clauses of P. A* valuation *of P wrt. a bilattice $\mathcal{L}$ is a mapping from* ground(P) *to truth values of $\mathcal{L}$. For any ground atom $A \in$* ground(P) *and valuation v, $v(A)$ denotes the truth value assigned to A by v.*

Let v and w be valuations wrt. $\mathcal{L}$. We define $(v \Delta w)$ as a mapping from ground literals to truth values of $\mathcal{L}$ as follows: $(v \Delta w)(A) = v(A)$ and $(v \Delta w)(\neg A) =$

$w(A)$. $(v \Delta w)$ *extends to more complex terms in the natural way:* $(v \Delta w)(B \vee C) = (v \Delta w)(B) \vee (v \Delta w)(C)$, $(v \Delta w)(B \oplus C) = (v \Delta w)(B) \oplus (v \Delta w)(C)$, *analogous for* $\wedge$ *and* $\otimes$.

As usual, we use a fixpoint operator to define the semantics of a program:

Definition 3 (Single Step Immediate Consequence Operator).
$\psi_P(v, w) = \{\langle A, u\rangle | A \leftarrow B \in \text{ground}(P) \wedge u = (v \Delta w)(B)\}$

ψ_P uses v to assign values to positive literals in rules and w to assign values to negative literals. Hence, we treat negation symmetrically. In our caching scenario we would use the local, cached and default information to initialize the valuations v and w. If no cached information is available, we start from the known true facts only. We have the following properties of ψ_P:

Proposition 1 (Properties of ψ_P [3]).
- $\psi_P(v, w)$ *is monotone wrt.* $\leq_k$ *in both* v *and* w;
- $\psi_P(v, w)$ *is monotone wrt.* $\leq_t$ *in* v *and*
- $\psi_P(v, w)$ *is anti-monotone wrt.* $\leq_t$ *in* w.

An operator $O(x)$ is anti-monotone, if $x_1 \leq x_2 \rightarrow O(x_2) \leq O(x_1)$. Obviously, $O^2(x)$ must be monotone. Since $\psi_P(v, w)$ is monotone in v, we can define

Definition 4 (Immediate Consequence Operator).
$\psi'_P(w)$ *is the least fixpoint of* $(\lambda v)\psi_P(v, w)$ *wrt.* $\leq_t$ *and* $\mathcal{L}$.

Similarly to ψ_P, we have the following properties of ψ'_P:

Proposition 2 (Properties of ψ'_P [3]).
- $\psi'_P(w)$ *is monotone wrt.* $\leq_k$ *and*
- $\psi'_P(w)$ *is anti-monotone wrt.* $\leq_t$.

$\mathcal{L}$ is a complete lattice, hence the first item guarantees, that fixpoints of ψ'_P exist and that there is a minimal one wrt. $\leq_k$. Obviously, as we still have w as parameter, $\psi'_P(w)$ can have multiple fixpoints.

Definition 5 (Stable Model).
Let P *be a program and* $\mathcal{L}$ *a complete bilattice. A fixpoint of* ψ'_P *wrt.* $\mathcal{L}$ *is called a* stable model *of* P. *The minimal stable model is called the* well founded model *of* P.

In contrast to the usual three or four valued definitions of the well founded and stable models, we can have true and false values with trust levels, but also $\top$ and $\bot$ values with trust levels. While this may seem a bit odd at first, it is very useful in our caching and trust setting: A $\top$ value with a trust level represents the maximally trusted information source, which is responsible for an inconsistency. We will use this idea in section 7 to resolve inconsistencies by dropping lowly trusted information. $\bot$ values with trust levels lead to a propagation of lower trust levels through the reasoning process, also in the presence of $\bot$ defaults or cache entries. As a result, we may come up with lowly trusted t, f or $\top$ values later. Intuitively, this reflects the assumption that the inferred value may be wrong, if the initial $\bot$ was already wrong in the cache or untrusted information source.

Theorem 1 (Complexity).
The data complexity of the well founded semantics wrt. $\mathcal{FOUR}-\mathcal{T}$ is polynomial, the combined complexity is EXPTIME.

Proof (sketch). We start from the known complexities of the well founded semantics in $\mathcal{FOUR}$. Our semantics is defined analogously to that in $\mathcal{FOUR}$. The only difference is that we no longer have a fixed logical bilattice. Therefore, we need to consider possible additional complexity for finding the supremum or infimum of two truth values. As our bilattices are built based on orders over a finite set of information sources, they must be finite. A finite bilattice $\mathcal{L}$ has finitely many edges $n \leq \mathcal{L}^2$. Assume we use a very basic algorithm to find a supremum (infimum): starting from both values to be compared, we follow all possible $\geq$-edges in the relevant order, until we find the other value. To do so, we need to follow less than $2n$ edges. Hence, the additional effort is polynomial.

Based on existing work for the well founded semantics on $\mathcal{FOUR}$ we can also define an operational semantics: The alternating fixpoint procedure proposed by Gelder [19] computes the two $\leq_t$-extremal stable models m_t and M_t of a program P, using the anti-monotonicity of ψ_P wrt. $\leq_t$. The well founded model of P is then obtained as $m_t \otimes M_t$ [3]. The alternating fixpoint procedure in turn gives rise to an implementation.

6 Extension towards OWL

In this section we extend $\mathcal{SROIQ}$, the description logic underlying the proposed OWL2 [8], to $\mathcal{SROIQ}-\mathcal{T}$ evaluated on a logical bilattice. The extension towards logical bilattices works analogously to the extension of $\mathcal{SHOIN}$ towards a fuzzy logic as proposed in [18]. Operators marked with a dot, e.g. $\dot{\geq}$ are the lattice operators described above, all other operators are the usual (two valued) boolean operators. For two valued operators and a logical bilattice $\mathcal{L}$ we map t to $max_t(\mathcal{L})$ and f to $min_t(\mathcal{L})$ to model that these truth values are absolutely trusted[2]. Please note that while we limit ourselves to $\mathcal{SROIQ}$ here, analogous extensions are possible for $\mathcal{SROIQ}(\mathcal{D})$ to support datatypes. Please also note that we do not include language constructs, which can be expressed by a combination of other constructs defined below. In particular, $Sym(R) = R^- \sqsubseteq R$ and $Tra(R) = R \circ R \sqsubseteq R$.

Definition 6 (Vocabulary).
A vocabulary $V = (N_C, N_P, N_I)$ is a triple where
- *N_C is a set of OWL classes,*
- *N_P is a set of properties and*
- *N_I is a set of individuals.*

N_C, N_P, N_I need not be disjoint.

[2] In $\mathcal{FOUR}-\mathcal{T}$ these would be f_∞ and t_∞, but we start with the general case.

A first generalization is that interpretations assign truth values from any given bilattice. In contrast, $\mathcal{SROIQ}$ is defined via set membership of (tuples of) individuals in classes (properties) and uses two truth values only.

Definition 7 (Interpretation).

Given a vocabulary V an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \mathcal{L}, \cdot^{\mathcal{I}_C}, \cdot^{\mathcal{I}_P}, \cdot^{\mathcal{I}_i})$ *is a 5-tuple where*

- $\Delta^{\mathcal{I}}$ *is a nonempty set called the object domain;*
- $\mathcal{L}$ *is a logical bilattice and Λ is the set of truth values in $\mathcal{L}$*
- $\cdot^{\mathcal{I}_C}$ *is the class interpretation function, which assigns to each OWL class $A \in N_C$ a function: $A^{\mathcal{I}_C} : \Delta^{\mathcal{I}} \to \Lambda$;*
- $\cdot^{\mathcal{I}_P}$ *is the property interpretation function, which assigns to each property $R \in N_P$ a function $R^{\mathcal{I}_P} : \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \to \Lambda$;*
- $\cdot^{\mathcal{I}_i}$ *is the individual interpretation function, which assigns to each individual $a \in N_I$ an element $a^{\mathcal{I}_i}$ from $\Delta^{\mathcal{I}}$.*

$\mathcal{I}$ is called a complete interpretation*, if the domain of every class is $\Delta^{\mathcal{I}}$ and the domain of every property is $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.*

We extend the property interpretation function $\cdot^{\mathcal{I}_P}$ to property expressions:

$$(R^-)^{\mathcal{I}} = \{(\langle x, y\rangle, u) | (\langle y, x\rangle, u) \in R^{\mathcal{I}}\}$$

The second generalization over $\mathcal{SROIQ}$ is the replacement of all quantifiers over set memberships with conjunctions and disjunctions over Λ. We extend the class interpretation function $\cdot^{\mathcal{I}_C}$ to descriptions as shown in table. 1.

Satisfaction of axioms in an interpretation $\mathcal{I}$ is defined in table 2. With $\circ$ we denote the composition of binary relations. For any function f, $dom(f)$ returns the domain of f. The generalization is analogous to that of $\cdot^{\mathcal{I}_C}$. Note that for equality of individuals, we only need two valued equality.

Satisfiability in $\mathcal{SROIQ} - \mathcal{T}$ is a bit unusual, because when using a logical bilattice we can always come up with interpretations satisfying all axioms by assigning $\top$ and $\bot$. Therefore, we define satisfiability wrt. a truth value:

Definition 8 (Satisfiability).

We say an axiom E is u-satisfiable *in an ontology O wrt. a bilattice $\mathcal{L}$, if there exists a complete interpretation $\mathcal{I}$ of O wrt. $\mathcal{L}$, which assigns a truth value $val(E, \mathcal{I})$ to E, such that $val(E, \mathcal{I}) \geq_k u$.*

We say an ontology O is u-satisfiable, if there exist a complete interpretation $\mathcal{I}$, which u-satisfies all axioms in O and for each class C we have $|\{a | \langle a, v\rangle \in C \wedge v \geq_t u\}| > 0$, that means no class is empty.

Now we define a special kind of satisfiability, which reflects our trust order:

Definition 9 (Trust Satisfiability).

Let $\mathcal{I}$ be a complete interpretation, O an ontology, which is composed from multiple data sources $\{S_1, ..., S_n\}$, and $\mathcal{T}$ a trust order over $\{S_1, ..., S_n\}$. Let source(E) denote the $\mathcal{T}$-maximal datasource, which axiom E comes from. O is trust satisfiable*, if there exists an $\mathcal{I}$, which satisfies O, such that for all axioms in table 2 and all $E \in O : val(E, \mathcal{I}) \geq_t t_{source(E)}$.*

$$
\begin{aligned}
\top^{\mathcal{I}}(x) &= \top_{yy}, \text{where y is the information source, defining} \top^{\mathcal{I}}(x) \\
\bot^{\mathcal{I}}(x) &= \bot_{yy}, \text{where y is the information source, defining} \bot^{\mathcal{I}}(x) \\
(C_1 \sqcap C_2)^{\mathcal{I}}(x) &= C_1^{\mathcal{I}}(x) \dot{\wedge} C_2^{\mathcal{I}}(x) \\
(C_1 \sqcup C_2)^{\mathcal{I}}(x) &= C_1^{\mathcal{I}}(x) \dot{\vee} C_2^{\mathcal{I}}(x) \\
(\neg C)^{\mathcal{I}}(x) &= \dot{\neg} C^{\mathcal{I}}(x) \\
(S^{-})^{\mathcal{I}}(x,y) &= S^{\mathcal{I}}(y,x) \\
(\forall R.C)^{\mathcal{I}}(x) &= \dot{\bigwedge}_{y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x,y) \dot{\rightarrow} C^{\mathcal{I}}(y) \\
(\exists R.C)^{\mathcal{I}}(x) &= \dot{\bigvee}_{y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x,y) \dot{\wedge} C^{\mathcal{I}}(y) \\
(\exists R.\text{Self})^{\mathcal{I}}(x) &= R^{\mathcal{I}}(x,x) \\
(\geq nS)^{\mathcal{I}}(x) &= \dot{\bigvee}_{\{y_1,\ldots,y_m\} \subseteq \Delta^{\mathcal{I}}, m \geq n} \dot{\bigwedge}_{i=1}^{n} S^{\mathcal{I}}(x,y_i) \\
(\leq nS)^{\mathcal{I}}(x) &= \dot{\neg} \dot{\bigvee}_{\{y_1,\ldots,y_{n+1}\} \subseteq \Delta^{\mathcal{I}}} \dot{\bigwedge}_{i=1}^{n+1} S^{\mathcal{I}}(x,y_i) \\
\{a_1,...,a_n\}^{\mathcal{I}}(x) &= \dot{\bigvee}_{i=1}^{n} a_i^{\mathcal{I}} = x
\end{aligned}
$$

Table 1. Extended Class Interpretation Function

$$
\begin{aligned}
(R \sqsubseteq S)^{\mathcal{I}} &= \dot{\bigwedge}_{x,y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x,y) \dot{\rightarrow} S^{\mathcal{I}}(x,y) \\
(R = S)^{\mathcal{I}} &= \dot{\bigwedge}_{x,y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x,y) \dot{\leftrightarrow} S^{\mathcal{I}}(x,y) \\
(R_1 \circ ... \circ R_n \sqsubseteq S)^{\mathcal{I}} &= \dot{\bigwedge}_{\langle x_1, x_{n+1} \rangle \in dom(S^{\mathcal{I}})} \dot{\bigvee}_{\{x_2,\ldots,x_n\}} \dot{\bigwedge}_{i=1}^{n} R_i^{\mathcal{I}}(x_i, x_{i+1}) \\
(Asy(R))^{\mathcal{I}} &= \dot{\bigwedge}_{x,y \in \Delta^{\mathcal{I}}} \dot{\neg}(R^{\mathcal{I}}(x,y) \dot{\wedge} R^{\mathcal{I}}(y,x)) \\
(Ref(R))^{\mathcal{I}} &= \dot{\bigwedge}_{x \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x,x) \\
(Irr(R))^{\mathcal{I}} &= \dot{\bigwedge}_{x \in \Delta^{\mathcal{I}}} \dot{\neg} R^{\mathcal{I}}(x,x) \\
(Dis(R,S))^{\mathcal{I}} &= \dot{\bigwedge}_{x,y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x,y) \dot{\rightarrow} \dot{\neg} S^{\mathcal{I}}(x,y) \\
(C \sqsubseteq D)^{\mathcal{I}} &= \dot{\bigwedge}_{x \in \Delta^{\mathcal{I}}} C^{\mathcal{I}}(x) \dot{\rightarrow} D^{\mathcal{I}}(x) \\
(a:C)^{\mathcal{I}} &= C^{\mathcal{I}}(a^{\mathcal{I}}) \\
((a,b):R)^{\mathcal{I}} &= R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \\
a \approx b &= a^{\mathcal{I}} = b^{\mathcal{I}} \\
a \not\approx b &= a^{\mathcal{I}} \neq b^{\mathcal{I}}.
\end{aligned}
$$

Table 2. Satisfaction of Axioms

Analogously we define consistency wrt. the knowledge order:

Definition 10 (Consistency).
Let $\mathcal{I}$ be a complete interpretation, O an ontology, which is composed from multiple data sources $\{S_1, ..., S_n\}$ and $\mathcal{T}$ a trust order over $\{S_1, ..., S_n\}$. Let source(E) denote the $\mathcal{T}$-maximal datasource axiom E comes from.

We say O is u-consistent, if there exists an $\mathcal{I}$, which assigns a truth value $val(E, \mathcal{I})$ to all axioms E in O, such that $u \leq_k val(E, \mathcal{I})$. $\mathcal{I}$ is called a u-model *of O.*

We say O is consistent, if there exist an $\mathcal{I}$, which assigns a truth value $val(E, \mathcal{I})$ to all axioms E in O, such that $\forall x : val(E, \mathcal{I}) \notin \{\top_x, \bot_x\}$. We say $\mathcal{I}$ is a model of O.

If $\mathcal{I}$ models O and trust satisfies O, $\mathcal{I}$ is called a trusted model *of O.*

Finally, we define entailment:

Definition 11 (Entailment).
O entails a $\mathcal{SROIQ} - \mathcal{T}$ ontology O' ($O \vDash O'$), if every model of O is also a model of O'. O and O' are equivalent if O entails O' and O' entails O.

The following theorem shows that we have indeed defined a strict extension of $\mathcal{SROIQ}$:

Theorem 2. *If $\mathcal{FOUR}$ is used as logical bilattice, $\mathcal{SROIQ} - \mathcal{T}$ is isomorphic to $\mathcal{SROIQ}$.*

Proof (sketch). From a model of a $\mathcal{SROIQ} - \mathcal{T}$ ontology O wrt. $\mathcal{FOUR}$, we can derive a model for the same ontology in $\mathcal{SROIQ}$ by doing the following steps:

- For each class C, replace $C(a) = t$ by $a \in C$ and $C(a) = f$ by $a \notin C$. Analogous for properties.
- As O is consistent, we do not have $\top$ and $\bot$ truth values in a model.
- The only connectives used in the ontology language are $\neg, \vee, \wedge$, these are equivalent to their boolean counterparts in $\mathcal{FOUR}$.
- As we only have one trust level, we can simply ignore it.
- replace "trust-consistent" in $\mathcal{SROIQ} - \mathcal{T}$ with "consistent" in $\mathcal{SROIQ}$. Analogous for satisfiability.

For a complete proof we need to show for every rule in tables 1 and 2 that we can transform it to the $\mathcal{SROIQ}$ form (wrt. $\mathcal{FOUR}$) by replacing conjunctions and disjunctions by quantifiers over set membership or inclusion.

7 Resolving Inconsistencies

Inconsistencies in ontologies often emerge, when ontologies are integrated from various sources using ontology modules, ontology mappings and similar mechanisms [12]. In this section we propose to use trust based reasoning for inconsistency resolution.

Definition 12 (Maximally Trust Consistent Interpretation).
We say an interpretation $\mathcal{I}$ is maximally trust consistent*, if it does not assign any artificial $\top$ values, i.e. $\top_{xy}$ with $x \neq y$. An ontology O is said to be maximally trust consistent, if it has an interpretation $\mathcal{I}$, which is maximally trust consistent.*

This means, a trust maximal interpretation can still be inconsistent, but such inconsistency then arises from information obtained from a single information source. We now define, how a maximally trust consistent ontology can be derived from any given ontology.

Various approaches for *repairing* inconsistent ontologies have been proposed. In most approaches, axioms are removed from the ontology until the rest is a consistent ontology (cf.[12]). There usually are multiple possible choices for axioms to remove. While this might not seem too bad in our example, consider a similar scenario involving a red traffic light and we accidentally remove $RedLightSituation \sqsubseteq BetterBreakSituation$. Here trust based reasoning comes into play. We use the trust level as input to a selection function, which determines axioms to be removed, a point left open in [12].

Definition 13 (Minimal Inconsistent Subontology).
Let O be an inconsistent ontology. A minimal inconsistent subontology *$O' \subseteq O$ is an inconsistent ontology, such that every $O'' \subset O'$ is consistent.*

Now trust maximal consistency is reestablished by iteratively removing all axioms with the lowest trust level from O', until the resulting ontology is trust maximal consistent. This captures the idea, that in the case of an inconsistency, humans tend to ignore lowly trusted information first.

If a trust maximal consistent ontology still is inconsistent, we need a different selection function. However, in this case already the *local* knowledge is inconsistent. A similar situation arises, if we have an inconsistency resulting from two incomparable information source. In the latter case, however, we can choose to discard information from both. Of cause, depending on the actual application, we can also use a more sophisticated selection function, choosing among axioms on the lowest trust level.

8 Related Work

Relevant related work comes from the fields of semantic caching, multi-valued logics — particularly based on logical bi-lattices — from belief revision and trust. The following works are closely related:

The term Semantic Caching refers to the caching of semantic data. Examples are such diverse topics as caching results of semantic webservice discovery [17], caching of ontologies [1] and caching to improve the performance of query engines [10]. These approaches have in common, that they discuss how to best do caching of semantic data. In this paper, we describe which additional information about

the reliability of knowledge we can infer, given a heterogeneous infrastructure containing semantic caches.

Katz and Golbeck propose to use trust levels obtained from the analysis of online social networks to prioritize default logics [11]. A trust level in a rule then is a global value. Here, we do not specify, how the trust order is determined, but assume it is supplied by the user. [11] is based on a two valued default logic, that means trust levels of inferred facts are not computed. In [15], rule based reasoning over annotated information sources is done to establish a trust relation between the provider and the requester of a resource. The focus, is on establishing a trust relation using fine grained negotiation, instead of determining trust in a statement.

Much work has been done about basing logical formalisms on bilattices (cf. [3], [9]). Most of these works, however propose a certain logic by manually designing a suitable bilattice, or discuss how a particular logic can be formalized using a bilattice. In contrast to these works, we do not propose a fixed bilattice or logic. Instead, we automatically derive logical bilattices for trust based reasoning. Hence, we propose a whole family of logics, which can automatically be tailored to the problem at hand.

Deschrijver et al. propose a bilattice based framework of handling graded truth [4]. They extend fuzzy logics, which has a single, continuous order $\leq_t$ towards bilattices which also have a "fuzzy $\leq_k$". While this is obviously closely related to ours, we propose to use possibly partial orders, instead of strict orders as in fuzzy logics. Additionally we show, how the logical framework an be used with rule based and description logics.

Using belief revision, a single, consistent world view is retained in the presence of contradictory information by discarding e.g. lowly trusted information. In contrast, paraconsistent reasoning, as applied here, limits the influence to inconsistencies to fragments of the knowledge base. An approach similar to ours, but based on belief revision is proposed in [7].

Other works can be considered orthogonal to our approach: Following Golbeck's categorization of trust [6], our approach deals with trust in content (vs. trust in people or services). Further, we provide means for computing trust. In contrast to existing systems (cf. [6], chap. 2), we allow to infer trust levels on the very fine level of axioms, instead of the usual level of documents. As our approach is agnostic to the actual trust order, e.g. social trust derived from social networks or P2P based algorithms for computing trust measures can be used to provide this order.

9 Conclusion

We have proposed an extension to the logical bilattice $\mathcal{FOUR}$, called $\mathcal{FOUR}-\mathcal{T}$, which allows to reason with trust levels. As bilattices are a basis for various logical formalisms, this allows to extend many languages with trust based reasoning.

We have started by applying the extension to the well founded semantics. We have re-formalized $\mathcal{SROIQ}$ to work on bilattices, so our extension is applicable

in both, open and closed world reasoning. As applications of $\mathcal{FOUR}-\mathcal{T}$ we have investigated caching and inconsistency resolution. We are sure that our mechanism can be a good component of a future Semantic Web trust layer. As part of our future work we will investigate additional applications. We will investigate the complexity of trust based reasoning with description logics and plan an implementation, extending existing reasoning engines.

Acknowledgements This work has been supported by the European project Lifecycle Support for Networked Ontologies (NeOn, IST-2006-027595).

References

1. B. Liang et al. Semantic Similarity Based Ontology Cache. In *APWeb*, 2006.
2. T. Berners-Lee. Semantic Web Road Map. http://www.w3.org/DesignIssues/Semantic.html [2008-05-12], 1998.
3. M. Fitting. Fixpoint Semantics for Logic Programming - A Survey. *Theoretical Computer Science*, 278(1-2), 2002.
4. G. Deschrijver et al. A Bilattice-based Framework for Handling Graded Truth and Imprecision. *Uncertainty, Fuzziness and Knowledge-Based Systems*, 15(1), 2007.
5. M. L. Ginsberg. Multivalued Logics: A Uniform Approach to Inference in Artificial Intelligence. *Computational Intelligence*, 4(3), 1992.
6. J. Golbeck. Trust on the World Wide Web: A Survey. *Web Science*, 1(2):131–197, 2006.
7. J. Golbeck and C. Halaschek-Wiener. Trust-Based Revision for Expressive Web Syndication. *Logic and Computation*, to appear, 2008.
8. B. C. Grau and B. Motik. OWL 2 Web Ontology Language: Model-Theoretic Semantics. http://www.w3.org/TR/owl2-semantics/ [2008-05], 2008.
9. P. Hitzler and M. Wendt. A uniform approach to logic programming semantics. *TPLP*, 5(1-2), 2005.
10. A. Kaplunova, A. Kaya, and R. Möller. Experiences with load balancing and caching for semantic web applications. In *Proc. of DL Workshop*, 2006.
11. Y. Katz and J. Golbeck. Social Network-based Trust in Prioritized Default Logic. In *Proc. of AAAI*, 2006.
12. P. Haase et al. A Framework for Handling Inconsistency in Changing Ontologies. In *Proc. of ISWC*, 2005.
13. A. Polleres. From SPARQL to rules (and back). In *Proc. of WWW*, 2007.
14. T. C. Przymusinski. The Well-Founded Semantics Coincides with the Three-Valued Stable Semantics. *Fundamenta Informaticae*, 13(4), 1990.
15. R. Gavriloaie et al. No Registration Needed: How to use Declarative Policies and Negotiation to Access Sensitive Resources on the Semantic Web. In *Proc. of ESWS*, 2004.
16. S. Schenk and S. Staab. Networked Graphs: A Declarative Mechanism for SPARQL Rules, SPARQL Views and RDF Data Integration on the Web. In *Proc. of WWW*, 2008.
17. M. Stollberg, M. Hepp, and J. Hoffmann. A Caching Mechanism for Semantic Web Service Discovery. In *Proc. of ISWC*, 2007.
18. U. Straccia. A Fuzzy Description Logic for the Semantic Web. In *Fuzzy Logic and the Semantic Web*. Elsevier, 2006.
19. A. van Gelder, K. Ross, and J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. *J. of the ACM*, 38(3), 1991.