Fine-Grained Diagnostics of Ontologies with Assurance

Stefan Rass, Fadi Al Machot and Kyandoghere Kyamakya
Alpen-Adria Universität Klagenfurt
Austria

1. Introduction

Description logics (DL) is a class of logics for knowledge modeling, which are derived from semantic networks. Essentially they are to be understood as accessible fragments of predicate logic of first order, allowing strong expressions to be formulated.
Other description logics permit strong (complex) expressions in a very compact representation. For description logics, there are special inputs and intuitive notation that facilitates the handling of them substantially.
Modeling in expert systems is very important, especially in systems within highly complex domains where spatial and temporal data needs to be modeled. The ontology model permits the use of a *reasoner* that can check definitions of the statements in the ontology for consistency against each other. It can also recognize which concepts are the best for which definitions, pursuing an optimal solution in terms of size, speed, etc. This is particularly helpful when dealing with multiple classes hierarchies, therefore expert systems permit creating complex concepts from definitions of simpler ones. So, an ontology is an engineering artifact or is a formal representation of knowledge as a set of concepts within a domain, it often includes classification based information and constraints capturing the knowledge about the domain (cf. Kohler et al. (2003)).
Rule-based systems are successfully applied across a lot of domains. The interest in ontologies has become stronger to develop a common rule base that could be computed by different rule engines. This effort has led to the development of several rule languages such as the *rule markup language* (RuleML), the *semantic web rule language* (SWRL), Metalog, ISO Prolog, and many others.
Beside the weaknesses of SWRL are the weaknesses of the *SPARQL protocol and RDF Query Language* (SPARQL), where RDF is the acronym for *resource description framework* (see World Wide Web Consortium (2010)), to query an ontology, which requires the query writer to understand the data structure of the RDF resources. This understanding can be derived from *eye parsing* where sometimes the RDF or OWL ontology are large and the human being is not able to follow any more. This can become a major obstacle when debugging or extending an existing ontology.
Other computing paradigms, such as constraint satisfaction, quantified boolean formulas (QBF), or first order logic (FOL), do not naturally offer the powerful expressive possibilities to define our knowledge database of to model the spatial and context models. In general the tasks posed in the constraint satisfaction paradigm are computationally intractable (NP-hard).

Thinking of real-time reasoning systems, the most challenging task is to provide decision support in competitive situations, which calls for fast and reliable decisions. Speaking about the quality of a decision, we can hardly hope to reliably forecast another entity's behavior during an interaction. This is due to different ontologies most likely residing in different agents, as well as other unexpected behavior of a human operator who decides to ignore the system's recommendation.

1.1 Assurance

In many situations, such as driving a car with an adaptive driving assistance system (ADAS), the situation at hand is not truly competitive, but equally sure not cooperative. Even if cooperative driving is intended by the individual assistance system (presuming those to be capable of vehicle-to-vehicle ad hoc networking), the ultimate decision lies with the driver, who can freely choose to obey or to ignore any recommendation. It follows that if we wish to *assure* a particular outcome tied to a recommendation, we have to consider the behavior of others as much as our own. Interestingly, concepts from game-theory do provide an elegant mean of having a guaranteed outcome regardless of what the other entities in the system do. These concepts have become known under the general term of *security strategies* (cf. Ghose & Prasad (1989)), and can effectively be carried over to the context of ontological reasoning.
For the modeling, we shall assume an ontology with reasoning capabilities as a mere function taking some input values (constituting the query) to some recommendation (by performing internal reasoning). For illustrative purposes, let us take adaptive driving assistance as an example, stressing it whenever ideas or concept shall be exemplified. Suppose that the ADAS has been queried for driving directions, provides us with three alternative recommendations for guiding us to a given destination as fast as possible.

1. drive straight at high speed.
2. turn left at the next junction, speed can be moderate.
3. turn over with high speed.

The first recommendation may arise when the route is correct, but not necessarily the shortest one. The second advise may be appropriate if a faster route can be reached from the next junction, and the third statement could indicate an even shorter route provided that one immediately turns around.
Having the additional objective that we seek to maximize safety, one can again use the ontology and reasoning capabilities to deduce the likelihood of an accident. Assume that the approaching traffic can either come at slow or fast speed, and is free to turn left, right or go straight. We cannot hope to correctly guess which one happens, so we seek the best possible outcome in this uncertain situation.
The ontology plays a twofold role in the above scenario, as it can be used to

1. derive a recommendation for the next action, and to
2. assess the consequences that such an action might have.

While ontologies can do both, they are usually not explicitly designed to yield the "best" recommendation. Certainty factors, fuzzy reasoning and many other such approaches have been introduced to handle uncertainty, but a quantitative optimization of responses is mostly beyond the scope. Designing an ontology to provide us with both,

- correct, and
- provably optimal,

recommendations is a hardly considered problem, and a solution proposal for it is the core contribution of this chapter. In the following, we discuss both problems separately, tackling the correctness issue with a refined form of diagnosis. Optimality of recommendation will be achieved with a post-processing step, to be described in section 4.

2. Ontological reasoning in practice

The most recent developments for ontology design is *ontology web language* (OWL), with its dialects OWL DL (particularly optimized for description logics) and OWL Lite. It has various sets of operators and it is based on different logical models which simplify the description of the concepts. The semantic web rule language is based on OWL DL, OWL Lite, and the Rule Markup Language (cf. Snidaro et al. (2008)). All rules are expressed in terms of OWL concepts (classes, properties, individuals). This means that rules can be used to extract new knowledge from existing OWL ontologies. Therefore complex rules must be transformed to the requirements of SWRL (cf. Matheus et al. (2005)). Also, there are no inference engines that fully support SWRL up to now.
The first step in building a context model is to specify the desired system behavior. The developer then lists a set of possible scenarios, where each scenario is a relationship between entities to be observed. The requirements for modeling information contexts (cf. Fuchs (2008)):

Applicability:	The model must restrict the domain of application.
Traceability:	The model must provide support for recording of provenance and processing of information.
Inference:	The model should include tools that permit the definition of new contextual categories and facts on the basis of low-order context.
Re-usability:	The model should allow re-usability in other independent modeling tasks.
Flexibility:	The model should not be easily changeable to extend the ontology.
Completeness:	The model should cover all relevant concepts and properties.
Redundancy:	The model should not contain a lot of defined instances that have the same properties.

Context reasoning extends context information implicitly by introducing deduced context derived from other types of context. It is a perfect solution to resolve context inconsistency. In the light of the key role that ontologies play in many applications, it is essential to provide tools and services to support users in designing and maintaining high quality ontologies. This calls for:

1. All named classes can be instantiated (i.e. there are no "abstract" classes)
2. Correctly captured intuitions of domain experts
3. Minimal redundancy and no unintended synonyms
4. Rich axiomatization and (sufficient) detail

Answer queries over ontology classes and instances, e.g.:

1. Find more general/specific classes
2. Retrieve individuals/tuples matching a given query

Context interpreters consist of context reasoning engines and context knowledge-bases (context KB). The context reasoning engines provide the inference services including inferring contexts, resolving context conflicts (basically a problem of *diagnostics*) and maintaining the consistency of context knowledge-bases. Different rules for consistency can be specified and

fed into the reasoning engines. The context KB provides the service that other components can query, add, delete or modify context knowledge stored in the context database (cf. Wang et al. (2004)).
However, the problem of the current rules and reasoning systems is that they do not offer high performance according the highly declarative way of the encoding of the problem and the execution time to find the "best" solution.
As an example the used semantic web rule language (SWRL) has no negation or disjunction and the arithmetic predicates which it offers are weak.

Obstacles in writing logical programs
Semantic web technology is widely used for reasoning and expert systems. Especially the use of SWRL to define relationships between classes and individuals in the ontology may cause major problems due to the long execution time of semantic web technology for querying the ontology (e.g. via SWRL and SPARQL). This is indeed a problem when creating practical expert systems ought to perform in real-time. Some existing paradigms as well suffer from negation as failure, which has been discussed by Matheus et al. (2005). Moreover, they not necessarily allow n-ary predicates within a rule.
Going away from SWRL, we still require constraints, negation as failure, and expressive representation that are decidable and permit reasoning (efficiently). In particular, some logical programming languages do not offer arithmetic operations "built-in", and numeric constraints can affect decidability. Since reasoning (like done in the Prolog-language for instance), is often a recursive procedure, its complexity can become exponential and thus infeasible for even a medium-size ontology.
Now, to avoid the weaknesses of the existing paradigms *answer set programming* (ASP) offers flexible and high performance reasoning. ASP is a declarative approach for modeling and solves search problems by representing them as logic programs.

2.1 Answer set programming

The importance of ASP lies in the fact that it provides meaning to logic programs with default negation "not". Many interesting applications exist in planning, reasoning about action, configuration, diagnosis, space shuttle control, spatial, temporal and probabilistic reasoning, constraint programming, etc.
The Technical University of Vienna (TU-Wien) hosts a the research group "knowledge based systems", whose members are running a project on "Answer Set Programming for the Semantic Web". The goal of this project is research towards methods for providing advanced reasoning services in the context of the Semantic Web, using declarative knowledge representation and reasoning techniques (see Eiter et al. (2005)).
A logic program in the language of *AnsProlog* (also known as *A-Prolog*) is a set of rules of the form:

$$a_0 \leftarrow a_1, \ldots, a_m, \text{not}\, a_{m+1}, \ldots, \text{not}\, a_n \tag{1}$$

where $0 \leqslant m \leqslant n$, each a_i is an atom of some propositional language and *not* represents *negation-as-failure*. A negation-as-failure literal (or *naf-literal*) has the form not a, where a is an atom. Given a rule of this form, the left and right hand sides are called the *head* and *body*, respectively. A rule may have either an empty head or an empty body, but not both. Rules with an empty head are called constraints, while those with an empty body are known as *facts*. A *definite rule* is a rule which does not contain naf-literals, and a *definite program* is solely composed of definite rules (cf. Baral et al. (2010)).

Let X be a set of ground atoms (i.e. all atoms constructed with the predicate in Herband base of a logic program). The body of a rule of the form (1) is satisfied by X if $\{a_{m+1}, \ldots, a_n\} \bigcap X = \emptyset$ and $\{a_1, \ldots, a_m\} \subseteq X$. A rule with a non-empty head is satisfied by X if either its body is not satisfied by X, or $a_0 \in X$. A constraint is satisfied by X if its body is not satisfied by X.
Since logic programs unify declarative and procedural representations of knowledge, one way to reason is by using Horn clauses, backward reasoning and selective linear definite clause (SLD) resolution. The *reduct* of a program is a possibility to generate answer sets. Given an arbitrary program, Π and a set of ground atoms, X, the reduct of Π w.r.t. X, Π^X, is the definite program obtained from the set of all ground instances of Π by:

1. deleting all the rules that have a naf-literal not a in the body where $a \in X$, and
2. removing all naf-literals in the bodies of the remaining rules.

A set of ground atoms X is an answer set of a program Π, if it satisfies the following conditions:

1. If Π is a definite program, then X is a minimal set of atoms that satisfies all the rules in Π.
2. If Π is not a definite program, then X is the answer set of Π^X. (Recall that Π^X is a definite program, and its answer set is defined in the first item (cf. Baral et al. (2010)).

The other advantage of ASP is that the order of program rules does not matter and the order of subgoals in a rule is also not relevant. For an example, if we have the famous problem "3-colorability", where we have a map and we want to check whether 3 colors (blue, yellow and red) are sufficient to color a map. A map is represented by a graph, with facts about nodes and edges.

```
vertex(a), vertex(b), edge(a,b).
```

Every vertex must be colored with exactly one color:

```
color(V,r) :- vertex(V), not color(V,b), not color(V,y).
color(V,b) :- vertex(V), not color(V,r), not color(V,y).
color(V,y) :- vertex(V), not color(V,b), not color(V,r).
```

No adjacent vertices may be colored with the same color

```
:- vertex(V), vertex(U), edge(V,U), col(C), color(V,C), color(U,C).
```

Of course, we need to say what colors are:

```
col(r).
col(b).
col(y).
```

After running this program we will get all possible coloring cases to color the whole map with three different colors. The other advantage of ASP that the order of program rules does not a matter and the order of subgoals in a rule does not a matter also.

2.1.1 Logic programming with ordered disjunction

Logic programming can be extended to allow us to represent new options for problems in the head of the rules. ASP gives us this ability by the way of ordered disjunctions. Using ASP under specific conditions reasoning from most preferred answer sets gives optimal problem solutions. Through logical programs with ordered disjunction (LPODs), such as normal logic programs we are able to express incomplete and defeasible knowledge through the use of default negation, they allow us to represent performances among intended properties of problem solutions which depend on the current context. It is possible to use the degree of satisfaction of a rule to define a preference relation on answer sets. We will present an alternative on game-theoretic grounds in section 4. Brewka (2002) defines a rule as having *degree 1* under the following condition: when A is an answer set of P, then A satisfies all rules of P. For example, let us plan a vacation: Normally you prefer to go to Mallorca but also to Stockholm (denoted by the preference relation $\prec$). Usually people prefer Mallorca over Stockholm, unless it is hot. If it is hot Mallorca is preferred over Stockholm. In summer it is normally hot, but there are exceptions. If it is winter, then Mallorca is no long considered (cf. Brewka (2002)).

$$\begin{aligned} Stockholm \prec Mallorca &\leftarrow \text{not}\, hot && \text{(rule 1)}\\ Mallorca \prec Stockholm &\leftarrow hot && \text{(rule 2)}\\ hot &\leftarrow \text{not}\, \neg hot, summer && \text{(rule 3)}\\ \neg Mallorca &\leftarrow rain && \text{(rule 4)} \end{aligned}$$

Without further information about the weather we obtain the single preferred answer set $A_1 = \{Stockholm\}$, there is no information that it might be *hot*, so rule 1 will determine preferences. A_1 satisfies all rules to degree 1. Now if we add a new fact *summer*, then the new answer set is $\{summer, hot, Mallorca\}$. If we add the literal *hot*, then the new answer set is $\{summer, \neg hot, Stockholm\}$. Finally, if we add the facts *summer* and *rain*. The single answer set is $\{summer, rain, hot, \neg Mallorca, Stockholm\}$, we see that it is not possible to satisfy all rules to degree 1. As in real life there are situations where the best options simply do not work out, there for LPODs are very well suited for representing problems where a certain choice has to be made. In general, using ASP we can optimize the solution we want to generate, we can improve the rules and define the constraints we are using to get the maximum optimization of the desired answer sets (solutions) (cf. Brewka (2002)). Assurance, as introduced in section 4, pursues similar goals.

2.1.2 Guess and check programs in answer set programming

Answer set programming (ASP) is widely used, expressing properties in NP (i.e. properties whose verification can be done in polynomial time), where answer sets of normal logic programs can be generated through solutions and polynomial time proofs for such properties. The solution of such problems can be carried out in two steps:

1. Generate a candidate solution through a logic program
2. Check the solution by another logic program (cf. Eiter & Polleres (2006))

However, it is often not clear how to combine Π_{guess} and Π_{check} into a single program Π_{solve} which solves the overall problem. If we simply take the union $\Pi_{guess} \vee \Pi_{solve}$ does not work, so we have to rewrite the program.
Theoretical results prove that for problems with Σ_2^P complexity, it is required that Π_{check} is rewritten into a disjunctive logic program $\grave{\Pi}_{check}$ such that the answer sets of $\Pi_{solve} = \Pi_{guess} \vee \grave{\Pi}_{check}$ yield the solutions of the problem, where $\grave{\Pi}_{check}$ emulates the inconsistency check for $\grave{\Pi}_{check}$ as a minimal model check, which is co-NP-complete for disjunctive programs. This becomes even more complicated by the fact that $\grave{\Pi}_{check}$ must not crucially rely the use of negation, since it is essentially determined by the Π_{guess} part. These difficulties can make rewriting Π_{check} to $\grave{\Pi}_{check}$ a formidable and challenging task (cf. Eiter & Polleres (2006)).
As an example, if we are talking about planning the problem to find a sequence of actions, which it takes the system from an initial state p_0 to a state p_n, where the states are changing over the time. Conformant planning looks for a plan L which works under all contingencies cases that may be caused by incomplete information about the initial state and/or nondeterministic actions effects which is Σ_2^P under certain restrictions (see Eiter & Polleres (2006)). We consider the problem of the"fire alarm", we have an alarm that there is a

fire in a building which is supported through a fire alarm system. Possible actions (states) of the system turn off the electricity and then to pump water. After just having turned off the electricity, it does not extinguish the fire, but only the pumping of water guarantees that it is really extinguished. Using the following guess and check programs $fire_{guess}$ and $fire_{check}$ respectively, we can compute a plan for extinguishing the fire by two actions, $fire_{guess}$ and $fire_{check}$, the program $fire_{guess}$ guesses all candidate plans $P = p_1, p_2, ..., p_n$ using time points for action execution,

```
fire_guess:
% Timestamps:
time(0).
time(1).
% Guess a plan:
turn_off(T) v -pump(T) :- time(T).
pump(T) v -pump(T) :- time(T).
% Forbid concurrent actions:
:- pump(T), turn_off(T).
```

while $fire_{check}$ checks whether any such plan P is conformant for the goal $g =$ not $extinguished(2)$ The final constraint eliminates a plan execution if it reaches the goal; thus, $fire_{check}$ has no answer set if the plan P is conformant.

```
fire_check:
% Initial state:
fired(0) v -fired(0).
% Frame Axioms:
fired(T1) :- fired(T),
             time(T),
             not -fired(T1),
             T1 = T + 1.
turned_off(T1) :- turn_off(T),
                  T1 = T + 1.
% Effect of turning off:
turned_off(T1) :- turn_off(T),
                  T1 = T + 1.
fired(T1) v -fired(T1) :- turn_off(T),
                          fired(T),
                          T1 = T + 1.
% Effect of pumping:
-fired(T1) :- pump(T),
              turn_off(T),
              T1 = T + 1.
% Check goal in stage 2 (constraint):
:- not fired(2).
```

The program $fire_{guess}$ generates the answer set $S = \{time(0), time(1), turn_{off}(0), pump(1)\}$ which corresponds to the (single) conformant plan $\{P = turn_{off}, pump\}$ for goal not $fired(2)$. Using the method $fire_{guess}$ and $fire_{check}$ can be integrated automatically into a single program $fire_{solve} = fire_{guess} \vee \hat{f}ire_{check}$ It has a single answer set, corresponding to the single conformant plan $P = \{turn_{off}, pump\}$ as desired.
With these examples in mind, we now turn to the problem of diagnosing such ontologies. As should have become evident by now, spotting an error in a large-scale program is a challenging task. We deliver a solution that is flexible and can be implemented with widely standard components. In particular, our proposal does not require substantial changes to an

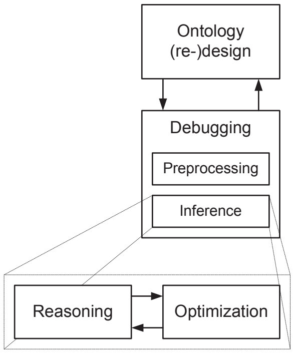

Fig. 1. Ontology design

existing diagnostic engine, so it can be seen as an "add-on" or refinement of a debugging system.

3. Fine-grained axiom diagnosis

In order to generalize diagnostic methods to fine-grained axiom diagnosis, we review the concepts behind model-based diagnosis. This prepares the ground for the diagnostic algorithm that concludes this section. The workflow when designing and debugging an ontology is briefly sketched in figure 1. Our focus in this section will be on debugging.
We assume the reader familiar with the resolution principle and predicate and propositional logic. For convenience, however, we will briefly review some basics and results before going into technical details (see (de Wolf & Nienhuys-Cheng, 1997, Chapter 2) for full details).
Let ψ be a first-order logic formula. An *interpretation* of ψ is an assignment of the variables to values from a given domain, and an assignment of predicates over this domain to truth-values, such that ψ becomes true. We write $\mathrm{lit}(\Sigma)$ for the set of literals appearing in either a formula Σ or a set Σ of formulas. For a set of formulas Σ, a *model* is an interpretation such that every $\psi \in \Sigma$ is true. Let Σ be a set of formulas and let ψ be a formula. We say that Σ logically implies ψ, writing $\Sigma \models \psi$, if every model of Σ is also a model of ψ. For sets of formulas, we write $\Sigma \models \Gamma$, if $\Sigma \models \psi$ for every $\psi \in \Gamma$.
A *clause* is basically a logical disjunction of literals. Let two clauses $C_1 = L_1 \vee \ldots \vee L_m$ and $C_2 = M_1 \vee \ldots \vee M_n$ be given and assume that there are two terms L_i and M_j that can be unified with each other such that $L_i = \neg M_j$. The *resolvent* of C_1, C_2 is the expression $C_1 \vee C_2$, having the terms L_i and M_j omitted. For the upcoming results, it is not necessary to fully introduce the concept of clauses and resolution, and we will confine ourselves to the following informal example of resolution: assume that we know that Peter plays either chess or football (clause C_1). In addition, we know that he does not play chess (clause C_2). We conclude that he must play football (the resolvent is thus $C_1 \vee C_2$ without the assertion of Peter playing chess).
For a clause C, we say that it can be *derived* from a set Σ, if there is a finite sequence of clauses $R_1, \ldots, R_n = C$ such that each clause R_i is either in Σ or a resolvent of two (previous) clauses R_j, R_k with $j, k < i$. In that case, we write $\Sigma \vdash_r C$.
Finally, we denote the empty clause as $\perp$, and note that a set of sentences is *inconsistent*, if it models the empty clause. Hence, writing $\Sigma \models \perp$ is equivalent to saying that a contradiction is derivable from Σ.

3.1 The general diagnosis problem

The theory of diagnosis as employed in this work is based on the seminal paper of Reiter (1987), with corrections made by Greiner et al. (1989). The method devised in the sequel proves axiom pinpointing to be a mere special case of standard diagnosis, except for some preprocessing. Hence, competing alternative approaches to the same problem, such as contrived by Schlobach & Cornet (2003) for instance, are technically interesting but come at the disadvantage of calling for intricate extensions to a diagnostic engine. Our approach comes at negligible additional computational cost, as we will need a parser for the axiom syntax (which cannot be avoided anyway).

To reason within an ontology is the process of deriving assertions about individuals from known facts and rules. Occasionally, we have certain facts and rules in ϕ that are undoubtedly correct and consistent, and we collect these in a set $B \subseteq \phi$, calling it the *consistent background theory*. These are explicitly excluded from the diagnostic process. The remainder $KB = \phi \setminus B$ is the set of facts and rules that are subject to the diagnosis. An ontology is usually verified by testing it against positive and negative test-cases.

Informally, a *diagnosis D* is a minimal set of components whose replacement will create a consistent ontology. This has been formally captured by Friedrich & Shchekotykhin (2005), and is clearly defined in

Definition 3.1 (Diagnosis (Problem)). *A* diagnosis problem *is a quadruple (KB, B, TC^+, TC^-), where KB is an ontology, comprising a set of facts and rules to be diagnosed, B is a consistent set of facts and rules (called the* background theory, *and TC^+ and TC^- are positive and negative test-cases. A* diagnosis *$D \subset KB$ is a minimal set (in terms of the subset-relation) of sentences such that there is an extension EX, where EX is a set of logical sentences added to KB, such that*

1. $\forall e^+ \in TC^+ : (KB \setminus D) \cup B \cup EX \cup \{e^+\} \not\models \bot,$
2. $\forall e^- \in TC^- : (KB \setminus D) \cup B \cup EX \cup \{e^-\} \models \bot.$

Rephrasing this definition, in essence it states that by replacing the components in the diagnosis D with the components in EX makes the ontology ϕ consistent with every positive test-case and inconsistent with every negative test-case.

Example: for illustration, consider the following example ontology:

$\mathcal{A}_1 : A_1 \rightarrow \neg A \wedge A_2 \wedge A_3$	$\mathcal{A}_2 : A_2 \rightarrow \neg D \wedge A_4$
$\mathcal{A}_3 : A_3 \rightarrow A_4 \wedge A_5$	$\mathcal{A}_4 : A_4 \rightarrow \forall s : F \wedge C$
$\mathcal{A}_5 : A_5 \rightarrow \exists s : \neg F$	$\mathcal{A}_6 : A_6 \rightarrow A_4 \wedge D$

The inconsistency is revealed by the following reasoning, starting with A_1 as a known fact: from $\mathcal{A}_1$, we can deduce (among others) the assertion that A_3 holds true. This one in turn implies the validity of A_4 and A_5 via rule $\mathcal{A}_3$. Yet A_4 tells that all individuals s enjoy property F, while A_5 implies that there is at least one individual that does *not* fall into the class F. This inconsistency is to be removed with the aid of diagnosis.

The problem tackled in the following is motivated by the (initially made) observation that the process will not remain that simple in a real-life ontology. Facing very complicated axioms, the diagnostic algorithms inspired by the work of Reiter (1987), essentially provide rather coarse-grained pointers towards the error. We shall improve on this.

Computing a diagnosis relies on the concept of *conflict sets* as introduced by de Kleer (1976) and adapted by Reiter (1987). Given a diagnosis problem (KB, B, TC^+, TC^-), a *conflict set* $C \subseteq KB$ is such that $C \cup B$ is inconsistent (either by itself or with any of the test-cases). A conflict is said to be minimal, if no subset of C is a conflict. Computing conflict sets can be done

with a divide-and-conquer approach, and we refer the reader to the QUICKXPLAIN-algorithm by Junker (2004) for that matter.
Returning to the example, for illustrative purposes, one conflict for the ontology is found as:

$$C = \{\mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5\}.$$

Removing the set C from the ontology makes everything consistent again. It is easy to construct examples where other conflicts can be found as well.
Computing a diagnosis then amounts to computing a hitting-set for the collection of all conflicts that exist within KB. This is basically theorem 4.4 in Reiter (1987). This is not surprising and easily justified intuitively: assume D to be a hitting-set for the set of minimal conflicts. Then retracting D from KB will reduce every conflict set by at least one element, thus destroying the conflict because those sets are minimal. Since this approach to diagnosis has seen widespread implementation and many appropriate systems are in place, we leave the details of the computation aside and concentrate on the problem of axiom pinpointing, which arises from a shortcoming of the so-far described method.

3.2 Pinpointing erroneous parts of an axiom

Despite the elegance of the existing diagnosis theory, its accuracy is limited to pointing towards entire axioms. Hence these can be rather complicated and lengthy, which makes the actual debugging process, i.e. finding the replacement EX (cf. definition 3.1) actually hard. Consequently, we would like to extend the scope of diagnosis to the smallest building blocks of the sentences within ϕ. This problem is known as *axiom diagnosis*. The trick is using the syntactical structure of a sentence to decompose a complicated axiom into a sequence of simple axioms representing only single logical connectives, quantifiers, etc. We will call this an *irreducible decomposition*, and the whole process of axiom pinpointing then boils down to diagnosing such an irreducible decomposition of the axiom to be debugged. We will expand the details in this section, strongly drawing from the work of Friedrich et al. (2006), whilst presenting further theoretical results that do not appear in this reference.
To ease technicalities in the following, we need two technical yet mild assumptions:

- axioms obey a non-ambiguous formal grammar G, such that every application of a production rule produces only *one* new logical connective or quantifier in each step of the derivation. We denote the language induced by the grammar G as $L(G)$, and assume $\mathcal{A} \in L(G)$ for every axiom $\mathcal{A}$, i.e. it is generated by G in a non-ambiguous manner. Although the ambiguity decision problem is undecidable for general grammars, those permitting disambiguation by lookahead tokens are provably unambiguous and thus suitable for our needs.
- the *open world assumption* states that a proposition is true if it can be proven, and assumed to be false if it can be disproved. Any non-present proposition remains unknown until proven to be true or false. In other words, reasoning under the open-world assumption demands first making choices before deducing anything.

Our first result formally captures the observation that the parse-tree of an axiom naturally yields a decomposition that only consists of axioms of the simplest possible form.
Let $\mathcal{A}$ be an axiom with its associated parse-tree. As an example, the axiom $A \to (B \wedge C) \vee D$ can be decomposed into

$$\zeta(\mathcal{A}) = \{A \to X_1 \vee X_2, X_1 \to X_3 \wedge X_4, X_3 \equiv B, X_4 \equiv C, X_2 \equiv D\}.$$

In this sense, an "atomic" axiom in the above sense, i.e. an axiom $\mathcal{A}$ for which the set $\zeta(\mathcal{A})$ is singleton, is said to be *irreducible*. For example, the formula $A \wedge B$ is irreducible, as it involves only a single connective between two literals. On the contrary, the formula $A \rightarrow \exists x : p(x) \wedge q(x)$ involves a quantifier, an implication and two unary predicates, and is therefore reducible. This formula can be decomposed into

$$\zeta(\mathcal{A}) = \{A \rightarrow X, X \rightarrow \exists x : R(x), R(x) \equiv P(x) \wedge Q(x), P \equiv p(x), Q \equiv q(x)\},$$

which is a set of irreducible axioms. For sets C of axioms, we define $\zeta(C) = \bigcup_{\mathcal{A} \in C} \zeta(\mathcal{A})$. Such a decomposition is called *irreducible*, if each of its elements is irreducible.
For example, the statement

```
p(A) :- (q(A,B), r(B)) ; s(A)
```

is irreducibly decomposed into the set ζ being

$$\begin{array}{llll} \{X_0 \rightarrow X_1,\ X_0 \equiv p(X_2),\ X_2 \equiv A, & X_1 \equiv X_3 \vee X_4,\ X_3 \equiv X_5 \wedge X_6,\ X_5 \equiv q(X_7, X_8), \\ X_7 \equiv A, \quad X_8 \equiv B, \quad X_6 \equiv r(X_9),\ X_9 \equiv B, & X_4 \equiv s(X_{10}), \quad X_{10} \equiv A\}. \end{array} \tag{2}$$

In the light of these considerations, the next result is immediately clear:

Lemma 3.1. *Let G be a non-ambiguous grammar satisfying the assumption above, and let $\mathcal{A}$ denote an axiom in $L(G)$. Then there is a unique set of axioms of the form $p(X, Y_1, \ldots, Y_n)$, where p is a logical connective or a quantifier, and $X, Y_1, \ldots, Y_n$ are literals.*

We denote the decomposition implied by the lemma 3.1 by $\zeta(\mathcal{A})$. It is quite obvious that $\zeta(\mathcal{A})$ is logically equivalent to $\mathcal{A}$, as we can recover $\mathcal{A}$ by back-substituting the literals, which is equivalent to stating that $\zeta(\mathcal{A}) \models \mathcal{A}$ and vice versa.
To ease notation, for logical implications or equivalences like $\mathcal{A} : X \rightarrow p(Y_1, \ldots, Y_n)$ or $\mathcal{A} : X \equiv p(Y_1, \ldots, Y_n)$, we write $lhs(\mathcal{A})$ to denote the set $\{X\}$, and $rhs(\mathcal{A})$ to collect all literals appearing on the right-hand side of $\mathcal{A}$, i.e. $rhs(\mathcal{A}) = \{Y_1, \ldots, Y_n\}$. Returning to the example above, we would have $lhs(X_0 \rightarrow X_1) = \{X_0\}$ and $rhs(X_0 \rightarrow X_1) = \{X_1\}$. For a set C of axioms, we set $lhs(C) = \bigcup_{\mathcal{A} \in C} lhs(\mathcal{A})$. The symbol $rhs(C)$ is defined analogously.
As a mere technical tool, we introduce a graphic representation of the structure of an axiom. It is closely related to the concept of parse-trees:

Definition 3.2 (structure graph). *Let $\zeta(\mathcal{A})$ be an irreducible decomposition of an axiom. The vertices of the* structure graph *$G^s_{\mathcal{A}}$ are given by $V(G^s_{\mathcal{A}}) = lhs(\zeta(\mathcal{A}))$, i.e. the vertices are the names of all axioms that appear in $\zeta(\mathcal{A})$. The graph has an edge between two axioms $\mathcal{A}_i$ and $\mathcal{A}_j$, if and only if $rhs(\mathcal{A}_i) \cap lhs(\mathcal{A}_j) \neq \emptyset$.*

Graphically, the decomposition (2) would look like shown in figure 2. The vertices of the structure graph are given by $lhs(\zeta) = \{X_0, \ldots, X_{10}\}$, reading off the left-hand sides from the decomposition (2). An arc is present between two vertices, if they appear on different sides of the corresponding axiom, i.e. X_0 and X_1 are connected because the decomposition contains the rule "$X_0 \rightarrow X_1$".
This provides us with an adequate tool for simplifying axioms before putting it to the diagnosis. Indeed, the structure graph's connectivity can be related to logical inconsistency:

Lemma 3.2 ((Rass, 2005, Proposition 5.3.3)). *Let C be a minimal conflict of an irreducible axiom decomposition $\zeta(C)$. Then the corresponding structure graph is connected.*

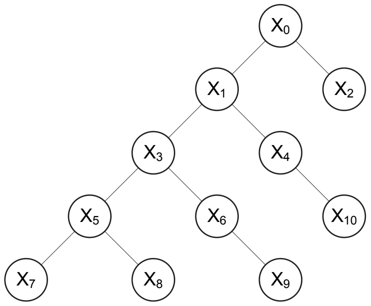

Fig. 2. Structure graph for decomposition ζ (eq. (2))

Proof. By construction, an irreducible axiom decomposition is tree-structured, so let X_0 denote the top-level literal. Assume the proposition to be false: Let C be a minimal conflict and let the corresponding structure graph be disconnected, having two non-empty connectivity components C_1 and C_2 ($C = C_1 \cup C_2$). Let S, S_1 and S_2 be standard forms of $\mathcal{B}, C_1$ and C_2, respectively. Then, by (de Wolf & Nienhuys-Cheng, 1997, theorem 5.20), we have

$$\mathcal{B} \cup C \models \perp \iff S \cup S_1 \cup S_2 \vdash_r \perp.$$

By definition, this is the case if and only if

$$S \cup S_1 \cup S_2 \vdash_r \{R_1, R_2\} \vdash_r \perp.$$

Since C is a minimal conflict, $\mathcal{B} \cup C_1$ is consistent as well as $\mathcal{B} \cup C_2$. Hence, we cannot have both R_1 and R_2 derivable from the same connectivity component, say C_1, for otherwise

$$S \cup S_1 \vdash_r \{R_1, R_2\} \vdash_r \perp$$

and therefore $\mathcal{B} \cup C_1 \models \perp$ by (de Wolf & Nienhuys-Cheng, 1997, theorem 4.39). This would mean that C_1 is a smaller conflict set than C, contradicting the minimality of C. So assume without loss of generality that

$$S \cup S_1 \vdash_r R_1 \text{ and } S \cup S_2 \vdash_r R_2. \tag{3}$$

By construction, there must be a sequence $R'_1, R'_2, \ldots, R'_n \equiv R_1$ with $R'_i \in S \cup S_1$ or R'_i a resolvent of $\left\{R'_p, R'_q\right\}$ for $p, q < i$. Since $\mathcal{B}$ is consistent, $S \nvdash_r R_1$ and there must be an R'_j so that a (leaf-)literal L_1 in S_1 occurs in R'_j, as no set of interior literals can solely contribute to the derivation of R_1. Hence we must have a path from X_0 to $X_k \equiv L_1$ by our assumptions on the grammar and the method of construction, for otherwise X_k (and therefore L_1) would not occur anywhere during the inference. The same argument applies for the derivation of R_2 from $\mathcal{B} \cup C_2$, so any inference giving $\{R_1, R_2\}$ necessarily induces paths from X_0 to the node $X_k \equiv L_1 \in \text{lit}(R'_1, R'_2, \ldots, R_1)$ and from X_0 to $X_l \equiv L_2 \in \text{lit}(R''_1, R''_2, \ldots, R_2)$. Yet $X_0 \in C_1$ (without loss of generality), and C_1 is not connected to C_2 in the structure graph, so the path from X_0 to X_l does not exist. This contradicts (3) and completes the proof. $\square$

Based on the previous result, we can state an interesting criterion that permits further minimizing conflict sets in terms of literals, even when they are already minimal in terms of set inclusion.

Proposition 3.3 ((Rass, 2005, Lemma 5.3.4)). *Let $\mathcal{K}$ be a knowledge-base and C be a minimal and irreducible conflict-set, and let $\mathcal{A} \in C$ be an axiom involving the literals $X, Y_1, \ldots, Y_n$. Then X can be removed without loosing the minimality or inconsistency if, and only if,*

$$X \in (rhs(C) \setminus lhs(C)) \cap \overline{lit(\mathcal{K})}.$$

Proof. ("if") Let $C = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be inconsistent, that is $B \cup C \models \bot$ for some consistent background theory B. Without loss of generality, we assume $\mathcal{A}_1 : p(\ldots, X, \ldots)$ to be the axiom that shall be simplified by removing X. Then it is clear that

$$B \cup (C \setminus \{\mathcal{A}_1\}) \cup \{\mathcal{A}_1\} \models \bot. \tag{4}$$

Since the literal X neither appears on the left-hand side of any axiom nor anywhere else in the knowledge-base, we do not have any specification or information on it. Thus, by the open-world assumption, equation (4) does especially hold if we assume an arbitrary but fixed value c for X. Let c be the neutral element for the operator p (which would be `false` if $p = \vee$ or `true` if $p = \wedge$, respectively). By setting $X \equiv c$ we get

$$B \cup [(C \setminus \{\mathcal{A}_1\}) \cup \{\mathcal{A}_1\}] \cup \{X \equiv c\} \models \bot,$$

which can be rewritten as

$$B \cup (C \setminus \{\mathcal{A}_1\}) \cup (\{\mathcal{A}_1\} \cup \{X \equiv c\}) \models \bot,$$

being equivalent to

$$B \cup (C \setminus \{\mathcal{A}_1\}) \cup \{\mathcal{A}_1^*\} \models \bot$$

with $\mathcal{A}_1^*$ being the axiom $\mathcal{A}_1$ having X removed.
If C is minimal, then so is $(C \setminus \{\mathcal{A}_1\}) \cup \{\mathcal{A}_1^*\}$, because removing $\mathcal{A}_1^*$ results in the set $C \setminus \{\mathcal{A}_1\}$, which is consistent due to the minimality of C.
("only if") we show that removing a literal in

$$\overline{\left((rhs(C) \setminus lhs(C)) \cap \overline{\mathrm{lit}(\mathcal{K})}\right)} = \overline{rhs(C)} \cup lhs(C) \cup \mathrm{lit}(\mathcal{K})$$

destroys either the conflict or the minimality: by the construction of an irreducible decomposition as used in lemma 3.1, the structure graph induced by C is a tree. Assume the existence of another axiom $\mathcal{A}_k : X \equiv q(\ldots) \in C$ and that the set is still a minimal conflict after having removed X from the right hand side of $\mathcal{A}_1$. However, if we do so, then the edge between $\mathcal{A}_1$ and $\mathcal{A}_k$ disappears and the tree breaks up into two non-connected components. By lemma 3.2, the resulting set is not a minimal conflict any more, thus contradicting our assumption. Furthermore, we cannot remove any literal $L \in \mathrm{lit}(\mathcal{K})$, because the construction employed by lemma 3.1 implies that all axioms having L appear on their right hand side are of the form $Y \equiv L$. This does not involve any operator so no further simplification is possible. □

Example: for illustrating the last result, take the set of three (inconsistent) rules

$$C = \{X_1 \to X_2 \wedge X_3 \wedge X_4, X_2 \to B, X_3 \to \neg B\}.$$

They form a minimal conflict set for some axiom decomposition. The literal X_4 appears on the right-hand sides of C, but not on the left-hand side, since we have $lhs(C) = \{X_1, X_2, X_3\}$

and $rhs(C) = \{X_2, X_3, X_4, B\}$. Since the literals X_i were auxiliary and introduced merely for the decomposition, the literal X_4 can be unhesitatingly deleted. This results in a simplified set

$$C_{\text{simplified}} = \{X_1 \to X_2 \wedge X_3, X_2 \to B, X_3 \to \neg B\},$$

which is still inconsistent for the same reasons as C is.

3.3 The axiom diagnosis algorithm

With all these preliminaries, the axiom pinpointing algorithm is essentially a standard diagnosis with preprocessing. It comprises two stages:

1. Decompose the given set of axioms $\mathcal{A}_1, \ldots, \mathcal{A}_n$ irreducibly into the set $KB = \bigcup_{i=1}^{n} \zeta(\mathcal{A}_i)$.
2. Run a standard diagnosis on KB.

It is known that diagnosis is a computationally hard problem (due to the need of reasoning for computing the conflict sets and the computation of hitting sets, which is an NP-complete problem in general Garey & Johnson (1979)). While the algorithm above has been stated previously in Friedrich et al. (2006), this preliminary work does not exploit the potential simplification provided by proposition 3.3, by instantly drawing away the attention from literals that cannot contribute to the inconsistency.
The idea of decomposing an axiom into (irreducible) sub-axioms permits controlling the granularity as well as the scope of the diagnosis most easily.

Controlling the granularity: We are free to define the production rules of the underlying grammar in such a way that more complicated axioms can be generated in one blow. A simpler, yet equally effective, approach is backsubstituting a selection of irreducible axioms, thus creating more complicated expressions, yet remaining structurally simpler than the original axiom. Returning to the previous example, one could backsubstitute some of the axioms in the decomposition (2) in order to obtain the (logically equivalent but less fine-grained) decomposition

$$\begin{aligned} \{X_0 \to X_1, \quad X_0 \equiv p(X_2), \quad X_2 \equiv A, \quad X_1 \equiv X_3 \vee X_4, \\ X_3 \equiv X_5 \wedge X_6, X_5 \equiv q(A, B), X_6 \equiv r(B), X_4 \equiv s(A)\}. \end{aligned}$$

Diagnosing the latter decomposition obviously provides us with less precise information than diagnosing the full decomposition (2).

Controlling the scope: If some parts of the axiom are correct beyond doubt, then we can simply shift these to the background theory $\mathcal{B}$ when running the diagnostic engine. Hence, the scope comprises all axioms in KB, explicitly excluding those in $\mathcal{B}$.

Example: Reiter's diagnostic algorithm, when applied to the example ontology given in section 3.1 returns the following three diagnoses:

1. $D_1 = \{A_3 \to A_4 \wedge A_5\}$
2. $D_2 = \{A_4 \to \forall s : F \wedge C\}$
3. $D_3 = \{A_5 \to \exists s : \neg F\}$

which just says that we may change one of three axioms in order to achieve a repair. Taking the first of these diagnoses D_1 and performing axiom-diagnosis on a decomposition of the axiom provides us with the following diagnoses (among others):

- $X_4 \equiv A_5$: This directly points at something wrong with the literal A_5 in the axiom. Indeed, A_5 permits deriving the expression $\exists s : \neg F$ which yields an inconsistency in connection with $\forall s : F$ (derived via A_4).
- $X_2 \equiv X_3 \wedge X_4$: This points at the $\wedge$-operator as possible flaw. This is correct, since if we replace the $\wedge$ by a $\vee$, we can easily avoid the inconsistency. Moreover, the diagnosis showed that both arguments of the $\wedge$ are necessary for the inconsistency, which is also correct as we have to be able to conclude $\mathcal{A}_4$ and $\mathcal{A}_5$ from $\mathcal{A}_3$ to be inconsistent.

Notice the appeal of this form of refined diagnosis, as the precise location of the error is actually marked with an "X". By then, it is up to the human engineer to do the repair properly.

4. Assurance

Suppose that an ontology upon receiving the query has presented us with a number of, say n_1, answers from which we can choose one. From the viewpoint of the ontology, each answer is logically correct, and in the absence of preferences, certainty factors, or other means of selection, we can only choose the best one subjectively. The goal of assurance is making this decision with a provable benefit. For that matter, we briefly introduce some elements from the theory of games, which will become handy when putting things together to a *reasoning engine with assurance*. The reader familiar with matrix games may safely skip section 4.1, and move on to section 4.2 directly.

4.1 Matrix-games

A (*non-cooperative n-person*) *game* $\Gamma = (N, PS, H)$ is a triple composed of a set $N = \{1, 2, \ldots, n\}$ of players being able to choose actions from their corresponding strategies within the set of sets $PS = \{PS_1, PS_2, \ldots, PS_n\}$. The i-th player, by taking action $s_i \in PS_i$ from his set PS_i of possible strategies, receives the payoff $u_i(s_i, s_{-i})$, where $u_i \in H$ and s_{-i} denotes the strategies chosen by i's opponents. The set H thus comprises the set of payoff functions for each player, i.e. $H = \left\{u_i | u_i : \times_{i=1}^{n} PS_i \to \mathbb{R}\right\}$. Although we will use the general definition here, our application use of game-theory will be with 2-player games, with player 1 being the user of the ontology, and player 2 being the whole set of remaining entities outside the user's scope.
A *(Nash-)equilibrium* is a choice $s^* = (s_1^*, \ldots, s_n^*)$ such that

$$u(s_i, s_{-i}^*) \leq u(s_i^*, s_{-i}^*) \quad \forall s_i \in PS_i$$

and for all $i \in N$, i.e. if any of the players solely chooses an action other than s_i^*, his revenue will decrease. It is easy to construct examples where equilibria are not existing among the pure strategies in PS_i. But, if strategies are understood as probabilities for taking certain actions during repetitions of the game, then Glicksberg (1952) has proven that equilibria exist for every game with continuous payoff functions. In that case, the payoff is averaged over the repetitions of the game, i.e. we consider the *expected payoff*. Strategies which are interpreted as probability distributions over the sets in PS are called *mixed strategies*, and we shall exclusively refer to these in the sequel. The set S_i consists of all mixed strategies over PS_i. A game is called *zero-sum*, if $\sum_i u_i = 0$, or in the two-person case, if $u_1 = -u_2$. The game is called *finite*, if the sets in PS are all finite. For a finite zero-sum game Γ_0, the average revenue under an equilibrium strategy is the *value* of the game, denoted as $v(\Gamma_0) = \max_x \min_y x^T Ay$.
How is this related to our above reasoning problem? Player 1 will be the user of the ontology, and player 2 will be the collection of all other agents in the system. The use of zero-sum games is convenient because it implicitly (and perhaps pessimistically) assumes the other agents to

cooperate with each other so that they can cause as much harm to player 1 as possible. Of course, this assumption is dramatic and most surely not correct, but as we seek to assure the quality of decisions against all scenarios, it turns out as a sharp worst-case scenario sketch. This is made explicit in the following result:

Proposition 4.1 (Rass & Schartner (2009)). *Let* $\Gamma = (N, PS, H)$ *with* $N = \{1, 2\}$, $PS = \{PS_1, PS_2\}$, *and* $H = \{x^T Ay, x^T By\}$) *be a bi-matrix game with game-matrices* $A \in \mathbb{R}^{|PS_1| \times |PS_2|}, B \in \mathbb{R}^{|PS_2| \times |PS_1|}$ *for player* 1 *(honest) and player* 2 *(adversary), respectively. Let* $\Gamma_0 = (N, PS, \{x^T Ay, x^T(-A)y\})$ *be the zero-sum game from player 1's perspective (i.e. player 2 receives the payoff* $-x^T Ay$*), and let* $v(\Gamma_0)$ *denote its value (i.e. average outcome under a Nash-equilibrium strategy in* Γ_0*). Then*

$$v(\Gamma_0) \leq (x^*)^T Ay^* \tag{5}$$

for all Nash-equilibria (x^*, y^*) *of the game* Γ.

The proof is by simply observing that player 2 can either play the zero-sum strategy of Γ_0 (in this case the assumption is valid and we get equality in (5)) or act according to his own wishes. In the latter case, he necessarily deviates from the zero-sum strategy and thus increases the expected revenue for player 1.

4.2 Reasoning games

The observation that a zero-sum game soundly models a worst-case scenario from one player's point of view (proposition 4.1) leads to a simple way of assuring the quality of a decision: whenever we are facing random behavior, proposition 4.1 permits calculating the worst-case distribution and provides us with a behaviorial rule so that we get an assured outcome under this worst imaginable scenario. This is what we call

> *Assurance:* when facing an uncertain situation, our recommendation should be such that it provides a guaranteed outcome, independently of how much the observed behavior deviates from the assumptions under which a decision was made.

Proposition 4.1 is the key to do this, and the process is made rigorous after the following
Example: let us return to the introductory example sketched in section 1.1. We now invoke the game-theory and the (properly debugged) ontology to get the best answer from the three candidates.
Recall that the recommendations were $PS_1 = \{(s, f), (l, m), (o, h)\}$, where

(s,f): drive straight (s) at high speed (f)

(l,m): turn left (l) at the next junction, speed can be moderate (m).

(o,h): turn over (o) with high speed (h).

The oncoming traffic can either be slow or fast, and is free to turn left, right or straight at the next junction. Hence, the set PS_2 is composed from each of possible combinations, i.e. $PS_2 = \{\text{slow}, \text{fast}\} \times \{\text{turn left}, \text{turn right}, \text{go straight}\}$, making up 6 combinations, which we abbreviate as pairs in $PS_2 = \{(s, l), (f, l), (s, r), (f, r), (s, s), (f, s)\}$.
Assume that the ontology can decide upon the likelihood of an accident for each combination in $PS_1 \times PS_2$. For example, if the recommendation is to drive straight at high speed, and the oncoming traffic goes left, then the likelihood of an accident is higher than it would be if the oncoming traffic goes straight too (considering which driver has to give priority). If the

ontology classifies the accident likelihood in discrete terms like "none" (value 0), "negligible" (value 1) and "significant" (value 2), then the resulting matrix A could look like

A	(s,l)	(f,l)	(s,r)	(f,r)	(s,s)	(f,s)
(s,f)	1	2	0	0	0	0
(l,m)	0	0	1	2	1	2
(o,h)	0	0	0	0	1	2

Solving this game for its equilibrium value gives $v(A) = 0.5$ with equilibrium strategies $x^* = (1/2, 1/2, 0)$ and $y^* = (1/2, 0, 1/2, 0, 0, 0)$. Observe that this indicates that – aiming at maximal safety – we should never be advised to turn over, and can take either of the remaining choices with equal probability. Following this rule, we end up having a less than negligible chance (the value is 0.5 and as such strictly less than the negligible-value 1) of having an accident.
This process can be repeated for different scenarios, but crucially hinges on the ontology to be *correct* and perform reasoning *efficiently*.
Each query to the system yields a different matrix-game with strategy sets PS_1, PS_2, with its own unique Nash-equilibrium solution (which can as well be pre-computed). The actual recommendation provided to the user is a random selection from PS_1, where the particular choice is drawn from the equilibrium profile. This solution, among the valid alternatives, is presented as the recommendation, along with possible alternatives that are not explicitly recommended but possible.
This method is indeed computationally feasible, as a large set of possible games along with their corresponding Nash-equilibria can be *pre-computed* and stored for later usage in a *game-database* (indeed, all we need is the strategy set PS_1 and the corresponding Nash-equilibrium; the game matrix itself can be discarded). If the input parameters are discrete, then the set of queries is finite and hence the number of such games remains tractable. This is even more substantiated by the fact that a human operator will most likely not enter more than a few parameters as well as these will not be entered at arbitrary precision. As humans tend to reason in fuzzy terms, any natural mapping of these to parameters of a query answering system will consist of a small number of inputs to specify within small ranges to get an answer.
The ontology is then used to select the particular game at hand and provide the best behavior under uncertain behavior of others. The overall workflow is depicted in figure 3. The only block to be further explained in this picture is the *sampler*, drawing the concrete recommendation from the set of possible ones according to the Nash-equilibrium. This is a trivial task, as it amounts to sampling from a discrete probability distribution. We refer the interested reader to Gibbons (1992) for a formal justification, as we will restrict our presentation to giving the sampling algorithm: assume that a discrete distribution is given by $(p_1, \ldots, p_n)$.

1. Generate a uniform random number x within the interval $[0, 1]$.
2. Find and return (as the result) the smallest integer k such that $x \leq \sum_{i=1}^{k} p_i$.

So the overall process of reasoning with assurance can be described in a sequence of simple steps, presupposing a logically consistent ontology:

1. Unless the current query is found in the set of pre-computed ones,
2. generate the set of candidate recommendations,

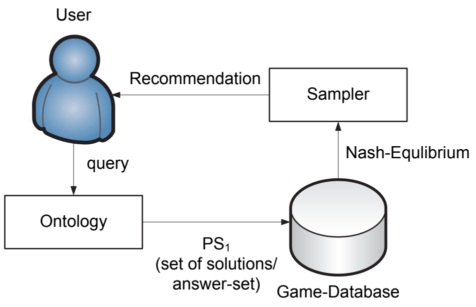

Fig. 3. Reasoning with game-theoretic assurance

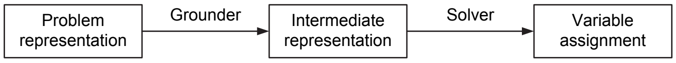

Fig. 4. Architecture of an ASP System (cf. Gebser (2008))

3. set up a game-matrix and solve for its equilibrium,
4. sample from the equilibrium profile and return the (so randomly selected) recommendation to the user.

From this process, it is evident that ontology and query optimization are major concerns, and therefore receive attention in the next section.

5. Maximizing reasoning performance

Optimization is the act of design and developing systems to take maximum advantage of the resources available. Ontology optimization formally defines the vocabularies in describing optimization methods and their configurations aiming to provide knowledge guidance in optimization selection and configuration Tao et al. (2004). Query optimization over an ontology consider the possible query plans to determine which of those plans will be most efficient. An ASP system usually consists of a grounder a grounder and a solver (see figure 4). First, a grounder translate A logic program in the language of AnsProlog s a non-ground problem description into a propositional program, which can be processed by the second component of the system, the solver (cf. Gebser (2008)).
There are algorithms that can be used for an effective optimization strategy for queries on knowledge databases query optimization.
One of the hardest problems in query optimization is the accurate estimation of the costs of alternative query plans. Cardinality estimation depend on estimates of the selection factor of predicates in the query. The estimation of the cardinality of a query usually is used to approximate the data transfer times of the result set, as part of the estimation of the total cost of executing a query (cf. Gebser (2008)). As an example a maximal complete query pattern path consists of a maximal query pattern path and a set of value constraints (cf. Shironoshita et al. (2007)). The total cardinality of such a path is obtained by calculating the product of the cardinality estimate of its maximal query path with all the value ratios for every variable in the query pattern path (cf. Shironoshita et al. (2007)).
In a highly distributed architecture where data in different locations connected through the internet, this is the most critical aspect of query execution time and the speed of the connections and the amount of data plays an important role.

A query optimization strategy is crucial to obtain reasonable performance over queries against ontology data models, especially if they are done over a highly distributed architecture. However, using the answer set programming paradigm, we are able to model the spatial and temporal context models and define the relationships between classes and individuals (rules) and generating the answer sets using advanced solvers that support binder splitting, backjumping and all other features. The statements can be optimized to find the maximal or the minimal answer set of the logic program. The statement can be weighted or not, therefore weights can be omitted. If there are several minimize maximize statements in the logic program the latter will be preferred. The difference between the performance of ASP solvers and other existed ontology query languages is clearly high.The estimation of the cardinality of a query usually is used to approximate the data transfer times of the result set, as part of the estimation of the total cost of executing a query.

6. Conclusion

In the light of the growing importance and complexity of nowadays ontologies, tools that support debugging of reasoning systems are as well an increasing demand. In many cases, particularly for legacy systems, complex rules and facts that contribute to an inconsistency can present themselves as mere "black-boxes" to the human engineer. Fine-grained diagnosis targets at turning such black-boxes into white-boxes by enhancing the accuracy of the diagnosis up to the smallest grammatical construct that the language permits. Even if an ontology is consistent, the user should not be overloaded with (logically correct) answers, without a tool for choosing among them. Besides taking the degree of satisfaction as a selection criterion, the concept of assurance yields answers that are provably optimal according to a user-definable goal. Hence, we can have the ontology give us the answer that will maximize the benefit for the user. The process is cyclic in the sense that assurance hinges on consistency and vice versa. Moreover, retrieving answers timely calls for optimization, which in turn can make rules and facts even more complicated, again calling for fine-grained diagnostics. This chapter is hoped to provide useful starting points to enter this (implicit) cycle at any point for future research.

7. References

Baral, C., Gelfond, G., Son, T. & Pontelli, E. (2010), 'Using answer set programming to model multi-agent scenarios involving agents' knowledge about other's knowledge', pp. 259–266.

Brewka, G. (2002), 'Logic programming with ordered disjunction', pp. 100–105.

de Kleer, J. (1976), 'Local methods for localizing faults in electronic circuits', *MIT AI Memo* 394.

de Wolf, R. & Nienhuys-Cheng, S. H. (1997), *Foundations of inductive logic programming*, Springer Verlag, Berlin, Heidelberg, New York.

Eiter, T., Ianni, G., Schindlauer, R. & Tompits, H. (2005), 'Nonmonotonic description logic programs: Implementation and experiments', pp. 511–527.

Eiter, T. & Polleres, A. (2006), 'Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications', *Theory and Practice of Logic Programming* 6(1-2), 23–60.

Friedrich, G., Rass, S. & Shchekotykhin, K. (2006), A general method for diagnosing axioms, *in* 'DX'06 - 17th International Workshop on Principles of Diagnosis', C.A. Gonzalez, T. Escobet, B. Pulido, Penaranda de Duero, Burgos, Spain, pp. 101–108.

Friedrich, G. & Shchekotykhin, K. (2005), A general diagnosis method for ontologies, *in* '4th ISWC', LNCS volume 3729, Springer.

Fuchs, S. (2008), *A komprehensive Knowledge Base for Context-Aware Tactical Driver Assistance Systems*, Shaker Verlag.

Garey, M. R. & Johnson, D. S. (1979), *Computers and intractability*, Freeman, New York.

Gebser, M, K. R. K. B. O. M. S. T. T. S. (2008), *Using gringo, clingo and iclingo*, UNI POTSDAM.

Ghose, D. & Prasad, U. R. (1989), 'Solution concepts in two-person multicriteria games', *Journal of Optimization Theory and Applications* 63(2), 167–189.

Gibbons, R. (1992), *A Primer in Game Theory*, Pearson Education Ltd.

Glicksberg, I. L. (1952), A further generalization of the Kakutani fixed point theorem, with application to nash equilibrium points, *in* 'Proceedings of the American Mathematical Society', Vol. 3, pp. 170–174.

Greiner, R., Smith, B. A. & Wilkerson, R. W. (1989), 'A correction to the algorithm in reiter's theory of diagnosis', *Artificial Intelligence* 41(1), 79–88.

Junker, U. (2004), QUICKXPLAIN: Preferred explanations and relaxations for over-contrained problems, *in* 'Proceedings of AAAI 04', San Jose, CA, USAs, pp. 167–172.

Kohler, J., Philippi, S. & Lange, M. (2003), 'SEMEDA: ontology based semantic integration of biological databases', *Bioinformatics* 19(18), 2420.

Matheus, C., Baclawski, K., Kokar, M. & Letkowski, J. (2005), 'Using SWRL and OWL to capture domain knowledge for a situation awareness application applied to a supply logistics scenario', pp. 130–144.

Rass, S. (2005), The diagnosis problem for axioms, Master's thesis, Universitaet Klagenfurt.

Rass, S. & Schartner, P. (2009), Game-theoretic security analysis of quantum networks, *in* 'Proceedings of the Third International Conference on Quantum, Nano and Micro Technologies', IEEE Computer Society, pp. 20–25.

Reiter, R. (1987), 'A theory of diagnosis from first prinicples', *Artificial Intelligence* 32(1), 57–95.

Schlobach, S. & Cornet, R. (2003), Non-standard reasoning services for the debugging of description logic terminologies, *in* 'Proceedings of IJCAI 03', Acapulco, Mexico, pp. 355–362.

Shironoshita, E., Ryan, M. & Kabuka, M. (2007), 'Cardinality estimation for the optimization of queries on ontologies', *ACM SIGMOD Record* 36(2), 13–18.

Snidaro, L., Belluz, M. & Foresti, G. (2008), 'Representing and recognizing complex events in surveillance applications', pp. 493–498.

Tao, F., Chen, L., Shadbolt, N., Xu, F., Cox, S., Puleston, C. & Goble, C. (2004), 'Semantic web based content enrichment and knowledge reuse in e-science', *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE* pp. 654–669.

Wang, X., Da Qing Zhang, T. & Pung, H. (2004), 'Ontology based context modeling and reasoning using OWL'.

World Wide Web Consortium (2010), 'Resource description framework (RDF)', `http://www.w3.org/RDF/`. (last accessed: 28th January, 2010).