Combining safe rules and ontologies by interfacing of reasoners

Jakob Henriksson
Fakultät für Informatik,
Technische Universität Dresden
Email: jakob.henriksson@tu-dresden.de

Jan Małuszyński
Institutionen för datavetenskap,
Linköpings universitet
Email: janma@ida.liu.se

***Abstract*— The paper presents a scheme for hybrid integration of rules languages with constraints languages including but not restricted to Description Logic-based ontologies. The proposed scheme is apt for logical rule languages such as Datalog, but also opens up for rules lacking logical semantics, e.g. the XML query and transformation language Xcerpt. To reason in the integrated language, we aim at re-using and interfacing existing reasoners for the component languages. Here we show how this can be accomplished for integrating Datalog and Xcerpt with OWL by interfacing XSB and an Xcerpt engine with a DL reasoner, respectively. Finally, we suggest ideas on how to improve reasoning performance by allowing for more frequent interaction between the component systems.**

I. Introduction

This paper addresses the issue of building the rule level on top of the ontology level of the Semantic Web tower [8]. As argued, e.g. in [26], applications need rules, which cannot be expressed in DL languages, such as OWL-DL. On the other hand, the rule languages should make it possible to integrate the structural knowledge provided by ontologies. There have already been several proposals in that direction, defining different specific languages integrating rules and ontologies (see e.g. [17], [14], [11], [15], [3], [20], [21], [23], [25]). The diversity of the languages seems to be unavoidable since different kind of applications will call for different languages integrating rules and ontologies. In contrast to the proposals mentioned above, our main objective is not to define a specific language integrating rules and ontologies, but a generic scheme for *hybrid* integration. A reasoner of an integrated language is then obtained by applying the scheme by interfacing existing reasoners of the component languages.

The idea of *hybrid* reasoning appeared already in [12], and was adopted, among others, in the well-known $\mathcal{AL}$-log work [10] on integrating Datalog and DL. It is also present in the CARIN work [22], even if this aspect is not explicitly stressed therein. In the context of the Semantic Web it is used in [11] for combining answer set reasoning with DL reasoning, and in [25] where theoretical issues of integration of disjunctive Datalog with OWL-DL are discussed.

This paper addresses the problem of hybrid integration of rules and ontologies in a more general framework of integrating rules with constraints expressed in a language of an external theory. The proposed framework applies to a class of rule languages with fixpoint semantics. We define a generic scheme for extending such rule languages by adding constraints in rule bodies. A fixpoint semantics of an extended language obtained in this way is formally defined by referring to the semantics of the components. The paper shows how to reason in the extended language by interfacing existing reasoners of the components instead of fully integrating them into a new dedicated system.

We illustrate the scheme by two example instances. First, we describe a reasoner for the integration of Datalog with OWL obtained by interfacing XSB Prolog [2] with any DIG [1] compliant DL reasoner (e.g. Racer [16]). Then, we describe how to integrate the rule-based XML query and transformation language Xcerpt [9] with OWL, which make possible semantic filtering of the XML documents obtained by Xcerpt queries.

When the rule language considered is Datalog and the constraint theory is expressed in a DL, our framework provides integrated languages that coincide with previous approaches (see Section V for more discussion). The main contributions of this paper is however a more general framework for integrating rule languages, not restricted to logical languages, with constraint theories (not necessarily a DL theory). The paper shows how the queries to an integrated KB can be answered by re-using existing reasoners of the component languages, specifically illustrated by a prototype system integrating Datalog with OWL using XSB Prolog [2] and a DL reasoner. While not currently implemented, we also describe the idea of how to achieve the same integration with Xcerpt as the rule language.

II. Preliminaries

The question addressed in this paper is how to combine a rule language with an ontology language so that reasoning in the integrated language can be done by interfacing reasoners of the component languages. This section formulates general requirements for the component languages and refers to the languages satisfying them.

A. Rules

We consider rules of the form

$$H \leftarrow B_1, \ldots, B_n$$

where, $n \geq 0$ and $H, B_1, \ldots, B_n$ are some primitive/atomic syntactic constructs (*atoms*) over a certain alphabet, including variables. As usual, we will call H the *head* of the rule

and $B_1, \ldots, B_n$ its *body*. Instances of a rule are created by *substitutions*, which map variables of the rule to terms. A rule with empty body (i.e. with $n = 0$) is sometimes called a *fact*. A rule will be called *safe* if all variables of the head appear in the body; thus safe facts are ground (i.e. variable-free). In this paper we only consider safe rules. To define the syntax of a specific rule language we thus have to define the syntax of the primitive rule constructs and the syntax of the terms. By a *rule program* we mean a finite set of rules.

The rules we consider can be used to derive new atoms from given ground atoms. For this a matching relation has to be defined between (possibly non-ground) body atoms and ground atoms. As a result of successful matching of body atoms and some given ground atoms, the variables of the body atoms become bound to ground terms. Due to the safeness assumption the resulting binding(s) applied to the head determines its ground instance(s) derived from the ground atoms matched by the body atoms. For every specific rule language a formally defined concept of matching makes it possible to associate an operator T_P on sets of ground atoms with every rule program P:

$$T_P(S) = \{H\theta \mid (H \leftarrow B_1, \ldots, B_n) \in P \text{ and } (B_1, \ldots, B_n) \text{ matches some } A_1, \ldots, A_n \text{ in } S \text{ with result } \theta\}$$

The operator is monotone, since the atoms which match a given pattern in a set S will also match it in any superset S' of S. Thus $T_P(S) \subseteq T_P(S')$ for any $S \subseteq S'$. The semantics of P can now be defined as the least fixpoint of T_P. We will call it the *standard model*[1] of P. Intuitively, the operator T_P reflects the mechanism for deriving ground atoms with rules of P.

Examples of rule languages in the discussed category are:

- Datalog (without negation), which is a decidable subset of FOL. The terms of Datalog are variables and constants. The atoms are built in a usual way from predicate symbols and terms. The semantics is based on syntactic matching (syntactic unification with ground terms). It is well-known that for a Datalog program P the least fixpoint of T_P is the least Herbrand model of P, which is the set of all ground atomic logical consequences of the rules of P considered as the formulae of FOL.
- A negation-free subset of the XML query and transformation language Xcerpt[2] [9]. Ground atoms of Xcerpt are called *data terms* and can be seen as abstraction of XML documents. A data term is either a constant or it is of the form $p[t_1, \ldots, t_n]$ or of the form $p\{t_1, \ldots, t_n\}$, $n \geq 0$ where p is a *label* and $t_1, \ldots, t_n$ are data terms. Intuitively, Xcerpt labels model XML tags. Thus, in contrast to predicate letters they do not have fixed arity and the number n of direct sub-terms t_i of a data term with label p may vary. The direct sub-terms of a data term may be ordered (which is indicated by square brackets) or unordered (which is indicated by braces). Body atoms of Xcerpt rules are called *query terms*. They are patterns matched against data terms and usually include variables, for which bindings to data terms are produced by successful matchings. The heads of Xcerpt rules are data terms with variables. The rule produces data terms by applying the bindings, obtained by matching of its body, to the head. The concept of matching is quite elaborate. A data term matched against a query term may produce more than one binding. There is no logical counterpart of the fixpoint semantics.

A common task to be solved by a rule reasoner is querying of the standard model of a given rule program. An atomic query is an atom A with variables. The answer is any substitution θ such that $A\theta$ is an element of the model.

As Datalog is a subset of Prolog, queries may be answered by Prolog systems based on SLD-resolution. The work presented in this paper uses XSB Prolog. Reasoning in the Xcerpt prototype[3], which is implemented in Haskell, is based on backward chaining and uses a special kind of unification.

B. Ontologies

In this paper, we consider ontologies formalized in Description Logics (DLs) [7], which are decidable subsets of first-order logic (FOL). The syntax of a DL is built over the distinct alphabets of *class names* $\mathcal{C}$ (also known as *concepts*), *property names* $\mathcal{R}$ (also known as *roles*) and individual names $\mathcal{O}$. Depending on the kind of DL, different constructors are provided to build class expressions (or briefly *classes*) and property expressions (or briefly *properties*). Intuitively, classes are used to represent sets of individuals of a domain and property expressions are used to represent binary relations over individuals. The names of the individuals are used to represent them and can be seen as logical constants. In Description Logics, it is often assumed that different names represent different individuals of the domain (*unique name* assumption).

By an *ontology* we mean a finite set of DL axioms of the form: $A \equiv C$ (*concept definition*), $C \sqsubseteq D$ (*concept inclusion*), $R \equiv S$ (*role definition*), $R \sqsubseteq S$ (*role inclusion*), $C(a)$ (*concept assertion*) and $R(a, b)$ (*role assertion*), where A is an atomic concept, C, D arbitrary concepts, R, S roles and a, b individuals. The axioms are thus of two different kinds and can accordingly be divided into two parts:

- a *T-Box* (terminology) consisting of concept (resp. role) definitions and inclusions;
- an *A-Box* (assertions) describing concept (resp. role) assertions relating to individuals.

Class expressions, property expressions and assertions can be seen as an alternative representation of FOL formulae. For example, class expression C where C is a class name corresponds to the FOL formula $C(x)$, and property expression R where R is a property name corresponds to the

[1]This terminology is justified by the fact that in the special case of Datalog, the least fixpoint of T_P is indeed a model in the sense of logic.

[2]The following presentation is oversimplified, neglecting many details. The objective is to give a minimal information needed to discuss integration of Xcerpt with OWL.

[3]http://www.xcerpt.org

FOL formula $R(x, y)$, where x and y are free variables. Similarly, expressions built with constructors can also be seen as FOL formulae. The inclusion axioms are equivalent to the universally quantified implications, e.g. $R \sqsubseteq S$, where R and S are property names corresponds to the formula $\forall x, y R(x, y) \rightarrow S(x, y)$. The assertions correspond to atomic formulae. Thus, the semantics of DLs is defined by referring to the usual notions of interpretation and model.

Due to the restricted syntax, Description Logics are decidable and are supported by dedicated reasoners.

Given an ontology Σ the reasoner is used to answer *queries*. The query languages supported by different reasoners may vary. For the work presented in this paper we are mostly interested in reasoning related to the A-Box of the underlying DL KB. Traditionally DL reasoners provide limited forms of querying on the A-Box, the most important service being the *instance check*, checking whether an individual is a member of some class. In our work we will need DL queries obtained by disjunction and/or conjunction of *basic conjunctive queries* defined as follows:

Definition A basic conjunctive query *is the existential closure of a formula of the form* $C(t)$ and $R(t_1, t_2)$ *where* C *is a concept,* R *is a role and* t, t_1, t_2 *are constants or variables, or the existential closure of the conjunction of such formulae*.

These are boolean queries, i.e. giving a *yes* or *no* answer. A query Q is to check if Q is a logical consequence of Σ. Only a few existing reasoners (see Section IV-A) answer conjunctive queries with additional syntactic restrictions. Disjunctive queries are usually not allowed.

There have been several proposals for ontology specification languages. A recent W3C standard OWL [24] comes in three versions, where OWL-DL is based on a highly expressive Description Logic and is supported by several reasoners.

III. Hybrid Integration of Safe Rules and External Theories

This section presents our framework for hybrid combination of rules and ontologies. Existing proposals are often restricted to rules with logical semantics. This makes it possible to provide logical semantics of the combined language and to prove that the proposed reasoning algorithm is sound and complete. The rule languages considered in this paper are assumed to have a fixpoint semantics. This does not exclude the cases of logical rule languages, like Datalog, but opens for languages for which a logical semantics may not be defined. Even for such rules there may be a practical motivation to integrate them with ontologies. For example, consider an XML database including culinary recipes. Each recipe lists ingredients using terminology of a food ontology. The ontology defines classes of products, e.g. a class of gluten-containing products. We may use Xcerpt rules to query the database for recipes, but to filter-out dishes not containing gluten we have to extend Xcerpt with ontology queries. This section outlines a systematic way for defining such extensions.

Let R be a rule program in a rule language and let Σ be a set of axioms in a first-order language L, to be called an *external theory*. In this paper we focus on external theories given by DL axioms encoded in OWL, but the discussion in this section is not restricted to this case. We assume that the languages share constants and variables while the predicate letters of the external theory are not in the alphabet of the rule language.

We define the language of extended rules by allowing formulae of L to be (optionally) added in the bodies of the rules of R. If a formula of L added to the body of a rule has free variables, they must also appear in the original rule. Thus an extended rule p has the form

$$H \leftarrow B_1, \ldots, B_m, C$$

where $H \leftarrow B_1, \ldots, B_m$ is a rule in R (called the *core rule* of p and denoted $p \downarrow$) and C, if present, is a formula of L called the *constraint*, whose free variables do not appear in the core.

A finite set P of extended rules will be called an *extended rule program*. By $P \downarrow$ we denote the set $\{p \downarrow | \ p \in P\}$. An extended rule p is said to be *safe* iff $p \downarrow$ is safe. We only consider safe rules. We assume that C is (implicitly) existentially quantified on all its free variables that do not appear in the core of the rule. Such a variable will be called *internal*. Notice, that due to the safety condition every free variable of a constraint that appears in the head must also appear in the body of the core rule.

Intuitively the constraints restrict the standard model of $P \downarrow$ by referring to the external theory Σ. Formally, we will consider *constrained* atoms of the form $A; C$ where A is a ground atom in R and C is a formula in L without free variables. A ground atom A is considered to be a constrained atom of the form $A; true$. By the *core atom* of a constrained atom $A; C$ to be denoted $(A; C) \downarrow$ we mean the atom A. The notation is extended to sets of constrained atoms: $S \downarrow = \{A \mid (A; C) \in S\}$.

We will first extend the definition of T_P to sets of constrained atoms:

$$\begin{aligned} T_P(S) = \{ & H\theta; (C\theta \wedge C_1 \wedge \ldots \wedge C_n) \mid \\ & (H \leftarrow B_1, \ldots, B_n, C) \in P \text{ and} \\ & \text{for some } A_1; C_1, \ldots, A_n; C_n \text{ in } S \\ & (B_1, \ldots, B_n) \text{ matches } A_1, \ldots, A_n \text{ with result } \theta \} \end{aligned}$$

It follows by this definition that $lfp(T_{P\downarrow}) = \{A \mid (A; C) \in lfp(T_P)\}$ since the extended operator does not use constraints for derivation of core atoms, but simply takes the conjunction of constraints as the associated constraints of the derived core atom. Thus the extended operator derives the same core atoms as the $T_{P\downarrow}$ operator but associates them with constraints. The semantics of the extended rule program P can now be defined as a subset of the standard model of $P \downarrow$ by referring to the associated constraints of the core atoms. Denote by C_A the disjunction of all constraints C such that the constrained atom $A; C$ is in the least fixpoint of T_P.

Definition *The* standard model of an extended rule program P over an external theory Σ *is defined as the set*

$$\mathcal{M}(P) = \{A \mid A \in lfp(T_{P\downarrow}) \text{ and } \Sigma \models C_A\}$$

Thus we restrict the standard model of $P \downarrow$ to those elements A for which the disjunction of all constraints associated with A by T_P is true in all models of the external theory Σ. In this way the semantics of the extended language is defined as a combination of the fixpoint semantics of the rule language with the logical semantics of the external theory. This applies to any particular rule language in the considered class and to any particular external theory. Obviously the membership problem for $\mathcal{M}(P)$ may be undecidable.

Consider the special case when the rule language component is Datalog (without negation). In this case extended rules are formulae of FOL. It can be proved that the standard model of an extended rule program P over Σ consists of atomic formulae that are logical consequences of the knowledge base $P \cup \Sigma$.

The least fixpoint of T_P can be computed by iterating T_P starting from the empty set. Due to the safety condition the core of any constrained atom produced by an iteration of T_P is ground and in the associated constraint all free variables are instantiated to some constants that appear in the program. Thus there is only a finite number of different constraint atoms that can be produced. An atom A is in $\mathcal{M}(P)$ iff it appears as a core of some constraint atoms in the least fixpoint of T_P and if the disjunctive constraint C_A is a logical consequence of the axioms of the external theory. Thus, if the theory is decidable, so is the membership problem for the standard model of any extended rule program over this theory. This applies in particular to combinations of Datalog with Description Logics, such as CARIN [22], restricted to safe extended rules. Note that our notion of a safe extended rule is different from the notion of a *role-safe* rule introduced in CARIN. Role-safe rules were introduced as a sufficient condition for decidability of the problem of whether or not a ground atom is a logical consequence of a given CARIN knowledge base.

In practice we want to query extended programs, e.g. by checking if a given ground atom A is in the standard model of P over Σ. This can be done by (1) constructing derivations of A and collecting the disjunction of the associated constructs (constructing C_A) (2) checking if C_A is a logical consequence of Σ. The reasoner of the rule language is able to query $P \downarrow$ with A. This is usually done by backward or forward rule chaining. However, it is not clear how to re-use the reasoner for P so that all associated constraints of A can be constructed. Problem (2) limits the approach to theories supported by sufficiently powerful reasoners.

In the following we show how the above mentioned problems can be solved for the special case of integrating Datalog with OWL, by interfacing XSB Prolog with a DL reasoner. We also sketch the idea of how the problems can be solved for integrating Xcerpt with OWL by interfacing an Xcerpt engine with a DL reasoner.

As discussed above, the query answering problem for an extended rule language may be undecidable, even though the outlined approach may be used for answering (some) queries. Well known examples of extended rule languages are:

- $\mathcal{AL}$-log [10] where the external axioms are in the language of the Description Logic $\mathcal{ALC}$ and Datalog rules are extended with constraints of the form $C(x)$ where C is a concept and x is a variable or a constant. Query answering in $\mathcal{AL}$-log is decidable. For every query the number of associated constraints is finite. The algorithm discussed in [10] uses SLD-resolution to construct them and a DL reasoner for checking validity of their disjunction wrt to a given theory.
- CARIN-$\mathcal{ALCNR}$ where the external axioms are in the language of the Description Logic $\mathcal{ALCNR}$ and Datalog rules are extended with constraints of the form $C(x)$ or $R(x, y)$ where C is a concept expression, R is a role expression and x, y are variables or constants. It should be noticed that CARIN rules may not be safe in our sense. It is only required that the variables of the head appear in the body, but their occurrence in non-constraint atoms is not assumed. Query answering in recursive CARIN is undecidable.

IV. Interfacing existing systems

This section describes applications of the proposed approach to interface existing systems.

In Section IV-A we briefly discuss what services existing DL reasoners usually support, since this is essential to solving the ontological constraints we address in this paper. In Section IV-B we give a first example instance of our scheme by describing how to reason in a language integrating Datalog with OWL by interfacing XSB Prolog with a DL reasoner. In Section IV-C we describe yet another example dealing with how to interface a reasoner for the XML query and transformation language Xcerpt with a DL reasoner in a similar fashion.

A. Ontology reasoners

In the rest of this paper, the only kind of constraints that we consider to appear in rules, are ontological. When we want to re-use existing reasoning engines for solving these constraints it is important to know what kind of constraints can be handled by these systems.

Checking (un)satisfiability of a KB is the most common reasoning procedure supported by existing DL systems and other services are usually reduced to it [7]. Some systems also provide more complex query languages, which are usually languages supporting conjunctive queries with some limitations on what kind of variables may be used (distinguished or non-distinguished) and how they may appear in the query. For a brief survey of such systems and the query languages they support, please refer to [5]. However, the constraints that we are required to solve, according to the description in Section III, are disjunctive and may include non-distinguished variables. The existence of non-distinguished variables is due

to our safety restriction. Thus, none of the existing systems supporting conjunctive query languages are sufficient for our purposes. Instead, we implement support for disjunctive queries (limited to concepts), which makes use of existing DL reasoners that are able to check (un)satisfiability of the underlying KB (see Section IV-B).

B. Combining Datalog with OWL

As mentioned in Section III we need a way to collect the constraints associated with a query A in order to interface a rule reasoner and a solver for the external theory. This collecting of constraints must be specific for every existing rule reasoner that is to be re-used in this hybrid context. In this section we show how this can be achieved for Datalog using a standard Prolog system (XSB Prolog) and also how we verify if the disjunction of the collected constraints is indeed a logical consequence of the associated theory. The external theory in this setting is a set of DL axioms represented as an OWL ontology.

We make use of the list-construct available in XSB Prolog to collect the constraints, i.e. atoms that are not to be solved by the rule reasoner. An extended rule program P is transformed into a corresponding program P' in the following manner. Every predicate is extended with a new parameter to represent the constraint associated with that atom. A rule fact has an empty body and is therefore associated with an empty list of constraints. E.g. a fact $p(a, b)$ is transformed into $p(a, b, [])$. The constraint atoms appearing in the body of a rule are moved into an additional head parameter and constructed as a list. E.g. the rule $p(X, Y) \leftarrow q(X, Y), R(X, Y), C(X)$, where R and C are ontological constraints, is transformed into $p(X, Y, [R(X, Y), C(X)|A]) \leftarrow q(X, Y, A)$. If there are more rule predicates in the body, the constraints of all of them are joined together into a single list using the list-construct *append* provided by XSB Prolog. We show this transformation on an example below.

The transformed program P' thus hides the external constraints in Prolog lists making sure that they are not evaluated by the rule engine. At the same time, the variables appearing in the constraints are properly grounded as expected when the rule is being evaluated. The program P' is executable in a Prolog system. Each derivation for a query A results in a conjunction of constraints. As already argued, we need to collect the constraints from all derivations of a query A and construct their disjunction. This is also how we treat the collected constraint list constructed by querying a transformed program P' (see example below).

The brief DL query language survey in Section IV-A informed us that the support for disjunctive queries is not well supported by existing DL systems. However, the theoretical solution of how to handle disjunctive queries (restricted to class expressions) is documented in literature (see e.g. [6],[18]). Most DL solvers implement satisfiability verification of a KB as the main reasoning service. All other services provided are reduced to the problem of checking satisfiability of the KB [7]. For example, to verify if the individual a is a member of the class C (*instance check*) the KB would be extended with the following axiom $\{a : \neg C\}$ whereupon satisfiability of the KB would be checked. The query $C(a)$ is a logical consequence of the KB if the extended KB is not satisfiable. A disjunctive query $C(a) \vee D(b)$ is solved by extending the KB with $\{a : \neg C, b : \neg D\}$ and again resolving to verifying (un)satisfiability [6]. Our safety condition does not enforce groundness of collected constraints but assures that no variable in a collected constraint is free. In particular, the internal variables of rules that appear in the collected constraints may be handled by the ontology reasoners discussed in Section IV-A as *non-distinguished* variables. We might have a constraint involving a non-distinguished variable like $C(X)$ where C is a concept and X a variable. In this case the KB is augmented with the axiom $\top \sqsubseteq \neg C$ whereupon (un)satisfiability of the extended KB is verified. A disjunctive query $Q_1 \vee \ldots \vee Q_n$ where the disjuncts are conjuncts of class expressions (what would be the result of evaluating a query wrt. a transformed Prolog program P' as described above) can be solved in the following manner [18]. The query is transformed into its conjunctive normal form (CNF). Each conjunct is a disjunction of class expressions which can be solved as described above. If all the conjuncts are held to be logical consequences of the underlying theory, then so is the original query.

We will look at an example (taken from [22] but slightly modified) where we show the steps performed by our prototype system to solve a query wrt. a hybrid knowledge base consisting of an extended Datalog rule-set and an OWL document.

$\mathbf{\Sigma}$

T-Box:

$European \sqcap American \sqsubseteq \bot$
$EuropeanAssociate := \exists Associate.European$
$AmericanAssociate := \exists Associate.American$
$NoFellowCompany := \forall Associate.\neg American$
$InternationalCompany := EuropeanAssociate \sqcup AmericanAssociate$

A-Box:

$\top(a), \top(high), InternationalCompany(b)$

Fig. 1. Company ontology described as DL axioms

$\mathbf{\Pi}$

r_1: price-in-usa(X,high) :- made-by(X,Y), NoFellowCompany(Y).
r_2: price-in-usa(X,high) :- made-by(X,Y), AmericanAssociate(Y), monopoly-in-usa(Y,X).
r_3: made-by(a,b).
r_4: monopoly-in-usa(b,a).

Fig. 2. Price rules

Given the query *price-in-usa(a,high)* wrt. the KB $\Sigma \cup \Pi$

Π' r_1: price-in-usa(X,high,[NoFellowCompany(Y)|A]) :-
made-by(X,Y,A).
r_2: price-in-usa(X,high,[AmericanAssociate(Y)|A]) :-
made-by(X,Y,A1),
monopoly-in-usa(Y,X,A2),
append(A1,A2,A).
r_3: made-by(a,b,[]).
r_4: monopoly-in-usa(b,a,[]).

Fig. 3. Transformed price rules

(Figure 1 and 2), the following steps are executed by our prototype system to solve the query.

1) The rule-base Π is transformed into Π' (Figure 3).
2) The query *price-in-usa(a,high,A)* is run by XSB Prolog wrt. the rule program Π'. The result as returned by XSB is:
$$A = [\ [c\_\_NoFellowCompany(c\_b)],\ [c\_\_AmericanAssociate(c\_b)]\ ]$$
where the prefix $c\_\_$ is simply used for convenience to refer to the specific underlying ontology.
3) Each sublist of the answer A correspond to a conjunction of class expressions. This disjunctive normal form (DNF) is turned into its CNF (one conjunct):
$$NoFellowCompany(b) \vee AmericanAssociate(b)$$
4) The underlying ontology is extended with the following two axioms:
$$b : \neg NoFellowCompany, b : \neg AmericanAssociate$$
and then a check is performed to see if the newly extended KB is satisfiable. If the extended KB is not satisfiable we conclude that the original query holds wrt. $\Sigma \cup \Pi$.

As explained in [22], the query *price-in-usa(a,high)* is true because b is either a member of the class $NoFellowCompany$ or the class $AmericanAssociate$ in all models of Σ (i.e. the constraint is a logical consequence of the KB).

This examples also gives a motivation as to why we need to collect the constraints from all derivations and construct a disjunctive constraint which then has to be verified wrt. the underlying KB. This can be seen since neither $NoFellowCompany(b)$ nor $AmericanAssociate(b)$ are logical consequences of Σ, but their disjunction is.

The prototypical system interfaces XSB Prolog with any DIG [1] compliant DL reasoner. DIG is a language for dealing with statements of DL. The Java library Jena[4] is used to handle the underlying ontology referenced by the rules. When solving the disjunctive DL queries, Jena is used to augment the KB with the additional axioms. Checking for satisfiability of the extended KB is also done via Jena to which a DIG compliant DL reasoner is connected. A well known DIG compliant reasoners used today is RACER [16].

[4]http://jena.sourceforge.net/

C. Combining Xcerpt with OWL

It is also interesting to extend non-logical rules languages with constraints in a similar way that was done with Datalog in Section IV-B. As hinted in Section III, one might want to extend an XML query language, such as Xcerpt, with ontological constraints in order to make use of domain knowledge from ontologies and thus be able to filter out certain unwanted results. E.g. to find all gluten-containing recipes in an XML database of recipes by referring to an ontology modeling food products.

Such an integration between Xcerpt and OWL was addressed in [27] and is in line with our integration scheme defined in Section III. However, in [27], an ad-hoc integration between the component reasoning engines was implemented and the Xcerpt engine used for this integration was altered to be usable in this new context. Since our aim is to reuse existing reasoning engines for the integration, we here show a way of achieving such an integration by re-using an unmodified Xcerpt engine. Thus, in this section our aim is to show a way to integrate Xcerpt with OWL by re-using an Xcerpt engine and a DL reasoner in a similar way to what was described for Datalog and OWL in Section IV-B.

The following Xcerpt data term, describing recipes and their ingredients, represents an XML database.

```
recipes [
 recipe [ name [ "Recipe 1" ],
   ingredients [
   ingredient [ name [ "sugar" ],
                amount [ attr { unit ["tbsp"] }, 3 ] ],
   ingredient [ name [ "orange" ],
                amount [ attr { unit ["unit"] }, 1 ] ]
                    ] ],
 recipe [ name [ "Recipe 2" ],
   ingredients [
     ingredient [ name [ "flour" ],
                  amount [ attr { unit ["dl"] }, 3 ] ],
     ingredient [ name [ "salt" ],
                  amount [ attr { unit ["krm"] }, 1 ] ]
                        ] ],
 recipe [ name [ "Recipe 3" ],
   ingredients [
     ingredient [ name [ "rice" ],
                  amount [ attr { unit ["dl"] }, 1 ] ],
     ingredient [ name [ "water" ],
                  amount [ attr { unit ["dl"] }, 2 ] ]
                      ] ],
 recipe [ name [ "Recipe 4" ],
   ingredients [
     ingredient [ name [ "barley" ],
                  amount [ attr { unit ["dl"] }, 3 ] ],
     ingredient [ name [ "salt" ],
                  amount [ attr { unit ["tbsp"] }, 1 ]
                      ] ] ]
]
```

When adding constraints to standard Xcerpt rules with the aim to filter out some uninteresting results, we need a way to extend the syntax of rules to accommodate this. To this aim we here adopt the syntax used in [27], i.e. use an additional Xcerpt rule construct *filter* where the ontological constraints are described. As in the Datalog case, the rules written in an extended syntax will be transformed into rules in syntax of the original rule language before being sent to the rule reasoner for processing.

The extended Xcerpt rule shown below is used to query an XML document *recipes.xml* (found above as an Xcerpt

data term) and to filter out recipes with no gluten-containing ingredient. The body of the rule finds names of recipes and ingredients in the XML document and binds them to the variables R and N, respectively. The *filter* construct reference the ingredient names (N) and requires that at least some ingredient of the recipe is found to be an instance of the concept *GlutenContaining* in the underlying ontology *http://www.owl.org/rec*. The head of the rule expresses that all recipes, for which the associated constraint holds, should be given as answers.

Π

```
GOAL
  out { resource { "file:result.xml" },
    results [ result [ all name [ var R ] ] ]
  }
FILTER
  in [ resource [ "http://www.owl.org/rec" ] as "ont"
    instance [
      ind   [ name [ var N ] ],
      catom [ name [ "ont#GlutenContaining" ]  ]
    ]
  ]
FROM
  in [ resource [ "file:recipes.xml" ],
    recipes [[
      recipe [[
        name [ var R ],
        ingredients [[ ingredient [[ name [ var N ] ]]
            ]]
      ]]
    ]]
  ]
END
```

DIG [1] is an XML-based language for communicating with DL reasoners. As an Xcerpt rule gives XML data as output, the constraint in the filter-construct is intentionally written as an Xcerpt data term, such that it can be used to output the ontological constraints in DIG syntax.

The idea is to transform the extended rule Π into an Xcerpt rule Π′, such that Π′ constructs the same answers as Π would, should the filter-construct be absent (i.e. answers produced by the rule Π ↓). The transformed rule must be a valid Xcerpt rule so that it can be processed by an existing Xcerpt engine. The core of Π (Π ↓) constructs the following answers, expressed as an Xcerpt data term:

```
results [
  result [ name [ "Recipe 1" ],
           name [ "Recipe 2" ],
           name [ "Recipe 3" ],
           name [ "Recipe 4" ] ] ]
```

However, the answers constructed by Π′ should (possibly) be associated with constraints that will have to be verified before the final answers can be returned to the user. If the associated constraint of a constructed answer is a logical consequence of the underlying ontology, then the answer is passed to the user, otherwise it is discarded. Thus, the rule Π above can be transformed into the rule Π′ below.

Π′

```
GOAL
  out { resource { "file:result.xml" },
    results [
      all result [
          name [ var R ],
          constraint [
            instance [
              ind   [ name [ var N ] ],
              catom [ name [
                "http://www.owl.org/rec#GlutenContaining"
                ] ]
        ] ] ] ]
  }
FROM
  in [ resource [ "file:recipes.xml" ],
    recipes [[
      recipe [[
        name [ var R ],
        ingredients [[ ingredient [[ name [ var N ] ]]
            ]]
      ]]
    ]]
  ]
END
```

The specified constraint (filter-construct) in Π was moved into the head of the rule of Π′, in a similar way to what was done for Datalog in Section IV-B, such that the constraints can be constructed. The following is the Xcerpt data term constructed by Π′:

```
results [
  result [
    name [ "Recipe 1" ],
    constraint [
      instance [
        ind [ name [ "sugar" ] ]
        catom [ name [
            "http://www.owl.org/rec#GlutenContaining"
            ] ] ] ] ],
  result [
    name [ "Recipe 1" ],
    constraint [
      instance [
        ind [ name [ "orange" ] ]
        catom [ name [
            "http://www.owl.org/rec#GlutenContaining"
            ] ] ] ] ],
  result [
    name [ "Recipe 2" ],
    constraint [
      instance [
        ind [ name [ "flour" ] ]
        catom [ name [
            "http://www.owl.org/rec#GlutenContaining"
            ] ] ] ] ],
  result [
    name [ "Recipe 2" ],
    constraint [
      instance [
        ind [ name [ "salt" ] ]
        catom [ name [
            "http://www.owl.org/rec#GlutenContaining"
            ] ] ] ] ],
  result [
    name [ "Recipe 3" ],
    constraint [
      instance [
        ind [ name [ "rice" ] ]
        catom [ name [
            "http://www.owl.org/rec#GlutenContaining"
            ] ] ] ] ],
  result [
    name [ "Recipe 3" ],
    constraint [
      instance [
        ind [ name [ "water" ] ]
        catom [ name [
            "http://www.owl.org/rec#GlutenContaining"
            ] ] ] ] ],
  result [
    name [ "Recipe 4" ],
    constraint [
      instance [
        ind [ name [ "barley" ] ]
        catom [ name [
            "http://www.owl.org/rec#GlutenContaining"
            ] ] ] ] ],
```

```
result [
  name [ "Recipe 4" ],
  constraint [
    instance [
      ind [ name [ "salt" ] ]
      catom [ name [
          "http://www.owl.org/rec#GlutenContaining"
          ] ] ] ] ] ]
```

In such a way, the output from Π' is basically a set of tuples, where each tuple consists of a constructed answer and an associated constraint. The output from the transformed rule needs to be parsed by a controlling system and the constraints needs to be verified using a DL reasoner. Since the constraints are already expressed in DIG syntax, this process is simplified if a DIG compliant reasoner is used, e.g. Racer or Pellet.

Two different constraints were constructed for the answer *Recipe 2*:

```
result [
  name [ "Recipe 2" ],
  constraint [
    instance [
      ind [ name [ "flour" ] ]
      catom [ name [
          "http://www.owl.org/rec#GlutenContaining" ]
          ] ] ] ],
result [
  name [ "Recipe 2" ],
  constraint [
    instance [
      ind [ name [ "salt" ] ]
      catom [ name [
          "http://www.owl.org/rec#GlutenContaining" ]
          ] ] ] ]
```

Thus, as explained in Section III, we should construct the disjunction of its constraints to verify it as a valid answer:

$$\Sigma \models GlutenContaining(flour) \vee GlutenContaining(salt)$$

where Σ is the underlying ontology. Since this constraint is assumed to hold ($flour$ is an instance of the concept $GlutenContaining$ in all models of Σ), the answer *Recipe 2* is valid and can be given as an answer to the original query. For the result *Recipe 1*, however, he following holds:

$$\Sigma \not\models GlutenContaining(sugar) \vee GlutenContaining(orange)$$

Thus, the answer *Recipe 1* is discarded. The following result should be returned to the user:

```
results [
  result [ name [ "Recipe 2" ],
           name [ "Recipe 4" ] ] ]
```

We have here tried to sketch an idea of how to integrate the rule language Xcerpt with OWL. The aim has been to transform an extended rule, which possibly includes ontological constraints, into a rule of the component language Xcerpt such that an unmodified Xcerpt engine can be re-used to work on the rule. The aim of the modified rule is two-fold, to construct answers to the original rule and at the same time to collect any ontological constraints to be verified by a DL reasoner.

Rule languages for the Semantic Web lacking a logical semantics, such as Xcerpt, have not traditionally been treated in integration schemes for rules and ontologies. We have here followed the ideas from [27] and have aimed to adapt it into our integration framework. Further investigations into practical treatment of Xcerpt as rule language in our framework is needed and is planned as future work.

In contrast to the integration in Section IV-B, this particular instance has not yet been implemented. However, it fits our integration framework and such a prototype is part of future work.

D. Outlook: Optimization strategies

As explained in Section IV-B, it is sometimes necessary to collect all possible constraints for a given query in order to prove its membership in the standard model of an integrated program. But, we might sometimes be able to reduce computation time by aborting rule reasoning in a preemptive fashion to verify some constraints, which possibly are sufficient to give an answer to the user. We will be referring to this technique as *eager interaction*. Three different strategies are available to us.

1) Conjunctive constraints For every found answer substitution to a query in a rule reasoner, there might be an associated conjunctive constraint, which needs to be verified using the constraint reasoner. In some cases, as in the example in Section IV-B, such a conjunctive constraint will not be sufficient to prove the query. Instead, a disjunctive constraint needs to be collected from several derivations of the query to possibly verify the query wrt. the underlying program. In other cases, such a conjunctive query will hold wrt. the external theory and will then be sufficient to give a correct answer to the query. We might exploit this fact and verify whether the associated constraint holds for every answer substitution found by the rule reasoner. Should this be the case, there is no need to collect a complete disjunctive constraint. Such an eager interaction scenario between the component reasoners would likely improve response time for users. This because the need to collect all possible constraints to a query in a rule reasoner before handing it over to the constraint solver, as done in Section IV-B, might rarely be needed in real applications.

2) Removing inessential constraints A conjunctive constraint collected by a rule reasoner wrt. a query to an integrated program might be insufficient to prove the query. However, as seen in Section IV-B, such a constraint might be important later as part of a disjunctive constraint constructed from several derivations of the query. On the other hand, a conjunctive constraint might also be found to be useless in such a case and should thus be discarded as soon as possible to save computation time in verifying the disjunctive constraint. This is the case if the negation of the conjunctive constraint is a logical consequence of the underlying external theory, i.e. $\Sigma \models \neg C$, where Σ is the theory and C is the constraint.

3) Eager checking disjunctive constraints Again, it might not always be the case that every constraint is needed in order to prove a query wrt. an external theory. When the query has not yet been proven, but at least two sets of constraints have been collected, their disjunction might be used to prove the query. While it is not known which constraints

will be required, eager interaction might reduce the number of constraints needed to be checked to prove a query wrt. a constraint domain.

While in some cases, eager interaction is likely to improve reasoning performance, it might also be costly due to the overhead of switching between the reasoning systems. Real experience with such eager interaction schemes is yet to be investigated and is part of future work.

V. Related Work

Our work extends the ideas of $\mathcal{AL}$-log [10] to a more general framework for hybrid integration of rules and constraint theories. An instance of the proposed framework is the prototype system of Section IV-B, an $\mathcal{AL}$-log style integration of Datalog and OWL-DL, based on re-use of existing reasoners.

In the language of extended rules supported by our prototype the constraint predicates are restricted to OWL concepts. Also in $\mathcal{AL}$-log constraints are restricted to concepts. This restriction is lifted in CARIN [22], where both concepts and roles are allowed as constraints in rules. The logic obtained in that way is undecidable in general. In contrast to CARIN our rules are safe, in which case allowing roles in constraints does not introduce undecidability.[5] Further extension of our prototype to such a subset of CARIN is possible, but would require a reasoner supporting disjunctive DL queries, where roles are allowed to appear.

Our approach is restricted to rules without negation and does not support non-monotonic reasoning. This facilitates definition of the semantics of an extended rule program as a restriction of the semantics of the underlying core rules. The core rules are assumed to have fixpoint semantics, and are not restricted to logical formulae. The approach can be easily extended to *stratified* rule programs with negation (for the notion of stratified logic program see e.g. [4]). This kind of negation is used, among others in Xcerpt. More advanced forms of negation and non-monotonic reasoning can only be handled by specific restrictions imposed on the considered rule languages. For example, some recent work on hybrid integration of rules and ontologies is based on *stable model* semantics or *answer set* semantics [13] for Datalog rules with negation. In the approach of [11], [21] the bodies of the extended rules may include ontology queries possibly locally modifying the A-Box of the ontology. The reasoning in the extended language can be done by re-using a rule reasoner supporting the stable model semantics and a DL-reasoner answering the DL queries. An extension and refinement of [11] is described in [21] which makes it possible to handle several DL KBs. Both [11] and [21] however do not take into account the issues discussed in [10], [22] with regards to completeness of the integration, whereas we do. *Safe hybrid knowledge bases* discussed in [25] provide a general formal framework for integrating DL ontologies and rules, where the rule languages considered include various subsets of disjunctive Datalog with stable model semantics. This approach allows DL predicates in the heads of rules, so that the interaction between DL and rules is more advanced than in our approach. The paper focuses on the semantic issues but sketches also a two-step algorithm for deciding satisfiability of a given hybrid KB, where one of the steps relies on standard DL reasoning and the other on standard search of stable model of Datalog rules.

Another approach to combining rules and ontologies does not stress hybrid reasoning but instead aims at defining a logical language extending DLs with rules. In such an approach there is no distinction between rule predicates and DL predicates, so that both the heads and the bodies of rules are built from concepts and roles. Examples of this approach include a decidable logic: the Description Logic Programs of [14] and an undecidable logic whose XML encoding is known as the Semantic Web Rule Language (SWRL) [17]. A closely related approach to [17] is a recent extension of OWL-DL with rules [23]. The integrated language is similar to the language in our prototype but we do not allow DL-predicates to appear in the heads of rules. Our safety condition is different from DL-safety of [23]. The latter requires that each variable of an integrated rule appears in a non-DL-atom in the rule body, while we only require that each variable in the head appears in a non-DL-atom of the body. The main distinction is that the query answering in [23] is done by using a compilation of the integrated program to disjunctive Datalog, while our prototype is a hybrid reasoner interfacing existing reasoners of the component languages.

The objectives of our work, aiming at re-using existing reasoners are not compatible with the language extension approach where a new reasoner has to be constructed for every new defined extension.

VI. Conclusions and Future Work

We presented a general scheme for combining various kinds of safe rules with various kinds of constraints. For a particular rule language with a fixpoint semantics and for a particular constraint language the scheme defines the syntax and the semantics of their composition. The language obtained in that way allows for specification of knowledge bases, consisting of extended rules and FOL axioms. Our scheme shows how reasoners of the underlying languages should be interfaced for querying the knowledge bases. The idea is to use the original rule reasoner on the cores of the extended rules while the constraints are to be checked by the original constraint reasoner. For this the rule reasoner has to be able to collect and instantiate the constraints associated with the core rules involved in reasoning. This feature is not supported by the existing rule reasoners but, as illustrated by our prototype, can sometimes be implemented by transformation of the source of the extended rules. To make existing rule reasoners applicable in our framework one should develop techniques for collecting constraints during their operation and for scheduling cooperation between the rule reasoner and the constraint solver.

We have in this paper considered a layered approach where a rule layer is put on top of an ontology layer. One can also

[5] Notice that our safety condition is different from that known as role-safeness, defining a decidable subset of CARIN.

consider several layers interleaving components of rules and ontologies. In the special case, when constraints are formulated in a DL, the A-Box can be specified by extended rules. To achieve such a multi-layering of rules and ontology components, one needs to define a component model describing how the components are interfaced with one another. From a software engineering perspective, this component model opens up the way of interoperability between various combinations of logical languages, for which type mappings between types in interfaces can be given. It would enable us to encapsulate the reasoners for the languages and to connect them via proxies, mapping the different data formats to each other. This would define a CORBA-like mechanism for logical languages, which is an inevitable interoperability mechanism for the future Semantic Web.

The prototype described in Section IV-B now only allows ontological constraints as *concepts*. It would be desirable to also support usage of *roles* in constraints as done in e.g. CARIN [22]. This is doable by plugging in already developed techniques for rolling-up of queries involving roles into queries which only contain concepts. Once this process is done, the constraint is rid of any roles and the techniques in Section IV-B can be used as described.

Another relevant topic is how to organize interaction of different constraint solvers when different kinds of constraints are used.

As the underlying rules of any extended rule program P are safe, the constraints in our approach are only used to restrict the finite model of $P \downarrow$. Admission of unsafe rules would enhance the expressive power of the extended rule languages. The family of extended rule languages obtained in that way would have a close relation to the CLP(X) family of constraint logic programming languages [19]. Clarification of this relation would allow for re-use of existing expertise of CLP in the Semantic Web.

ACKNOWLEDGEMENT

The authors are very grateful to Boris Motik, Michael Wessel and Birte Glimm for their helpful comments on the state of query languages of existing DL reasoners and to Wlodek Drabent and Artur Wilk for stimulating discussions on integration of Xcerpt and OWL.

This research has been co-funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. `http://rewerse.net`).

REFERENCES

[1] DIG Interface. WWW Page, 14 March 2006. Available at `http://dig.sourceforge.net/`.

[2] XSB. WWW Page, 14 March 2006. Available at `http://xsb.sourceforge.net/`.

[3] G. Antoniou. Nonmonotonic rule systems using ontologies. In *Proc. Intl. Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002.

[4] K. Apt and R. Bol. Logic programming and negation: A survey. *J. of Logic Programming*, 19/20:9–71, 1994.

[5] U. Assmann, J. Henriksson, and J. Maluszynski. Combining safe rules and ontologies by interfacing of reasoners. In *Proc. of Princples and Practises of Semantic Web Reasoning*, 2006.

[6] F. Baader, H.-J. Bürckert, B. Hollunder, W. Nutt, and J. H. Siekmann. Concept logics. Technical Report RR-90-10, 1990.

[7] F. Baader, D. Calvanese, and D. McGuiness(et.al.), editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[8] T. Berners-Lee. Semantic web tower. WWW Page. Available at `http://www.w3.org/2001/09/06-ecdl/slide17-0.html`.

[9] F. Bry and S. Schaffert. The XML query language Xcerpt: Design principles, examples, and semantics. In *Web and Databases, Proc of the 2nd Int. Workshop*, LNCS2593. Springer Verlag, 2002.

[10] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-Log: Integrating datalog and description logics. *Intelligent Information Systems*, 10(3):227–252, 1998.

[11] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. In *Proc. of the International Conference of Knowledge Representation and Reasoning (KR'04)*, 2004.

[12] A. Frisch and A. Cohn. Thoughts and afterthoughts on the 1988 workshop on principles of hybrid reasoning. *AI Mag.*, 11(5):77–83, 1991.

[13] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[14] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of 12th International Conference on the World Wide Web*, 2003.

[15] Guido Governatori. Defeasible Description Logics. In *RuleML*, pages 98–112, 2004.

[16] V. Haarslev and R. Möller. Description of the RACER system and its applications. In *DL2001 Workshop on Description Logics, Stanford, CA*, 2001.

[17] I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM, 2004.

[18] I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 399–404. AAAI Press / The MIT Press, 2000.

[19] Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[20] Kewen Wang, David Billington, Jeff Blee and Grigoris Antoniou. Combining Description Logic and Defeasible Logic for the Semantic Web. In *RuleML*, pages 170–181, 2004.

[21] Kewen Wang, Grigoris Antoniou, Rodney W. Topor and Abdul Sattar. Merging and Aligning Ontologies in dl-Programs. In *RuleML*, pages 160–171, 2005.

[22] A. Levy and M. Rousset. CARIN: A representation language combining horn rules and description logics. *Artificial Intelligence 104(1 2):165 209*, 1998.

[23] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. *J. of Web Semantics*, 3:41–60, 2005.

[24] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL web ontology language semantics and abstract syntax. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-semantics/`.

[25] R. Rosati. Semantic and computational advantages of the safe integration of ontologies and rules. In F. Fages and S. Soliman, editors, *Principles and Practice of Semantic Web Reasoning*, LNCS3703, pages 50–64. Springer Verlag, 2005.

[26] S. Staab(ed.). Where are the rules. *IEEE Intelligent Systems*, pages 76–83, September/October 2003.

[27] H. E. Svensson and A. Wilk. XML Querying Using Ontological Information. In *Proc. of Princples and Practises of Semantic Web Reasoning*, 2006.