Reasoning with Rules and Ontologies*

Thomas Eiter[1], Giovambattista Ianni[1], Axel Polleres[2],
Roman Schindlauer[1], and Hans Tompits[1]

[1] Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter, ianni, roman, tompits}@kr.tuwien.ac.at
[2] Universidad Rey Juan Carlos, 28933 Móstoles, Spain
axel@polleres.net

Abstract. For realizing the Semantic Web vision, extensive work is underway for getting the layers of its conceived architecture ready. Given that the Ontology Layer has reached a certain level of maturity with W3C recommendations such as RDF and the OWL Web Ontology Language, current interest focuses on the Rules Layer and its integration with the Ontology Layer. Several proposals have been made for solving this problem, which does not have a straightforward solution due to various obstacles. One of them is the fact that evaluation principles like the closed-world assumption, which is common in rule languages, are usually not adopted in ontologies. Furthermore, naively adding rules to ontologies raises undecidability issues. In this paper, after giving a brief overview about the current state of the Semantic-Web stack and its components, we will discuss nonmonotonic logic programs under the answer-set semantics as a possible formalism of choice for realizing the Rules Layer. We will briefly discuss open issues in combining rules and ontologies, and survey some existing proposals to facilitate reasoning with rules and ontologies. We will then focus on *description-logic programs* (or *dl-programs*, for short), which realize a transparent integration of rules and ontologies supported by existing reasoning engines, based on the answer-set semantics. We will further discuss a generalization of dl-programs, viz. HEX-*programs*, which offer access to different ontologies as well as higher-order language constructs.

1 Introduction

For the realization of the Semantic Web, the integration of different layers of its conceived architecture is a fundamental issue. In particular, the integration of *rules* and *ontologies* is currently under investigation, and many proposals in this direction have been made. They range from homogeneous approaches, in which rules and ontologies are combined in the same logical language (e.g., in SWRL and DLP [31, 24]), to hybrid approaches in which the predicates of the rules and the ontology are distinguished and suitable interfacing between them is facilitated, like, e.g., [18, 14, 59, 30] (see [4] for

* This research has been partially supported by the European Commission within the FP6 project REWERSE (IST 506779, http://rewerse.net), by the Austrian Science Fund (FWF) project P17212-N04, and by the CICyT of Spain project TIC-2003-9001-C02.

a survey about such approaches). While the former approaches provide a seamless semantic integration of rules and ontologies, they suffer from problems concerning either limited expressiveness or undecidability, because of the interaction between rules and ontologies. Furthermore, they are not (or only to a limited extent) capable of dealing with ontologies having different formats and semantics (like, e.g., RDF Schema and OWL DL, which have some semantic incompatibilities) at the same time. This can be handled, in a fully transparent way, by approaches which keep rules and ontologies separate. Here, ontologies are treated as external sources of information, which are accessed by rules that also may provide input to the ontologies. In view of well-defined interfaces, the precise semantic definition of ontologies and their actual structure does not need to be known. This in particular facilitates ontology access as a Web service, where also privacy issues might be involved (e.g., a customer taxonomy in a financial domain).

In this paper, we shall consider reasoning with rules and ontologies from the *answer-set programming* (ASP) [6] perspective. The latter is nowadays a general term for a powerful knowledge representation (KR) and declarative programming paradigm which includes many language features from nonmonotonic logics, as well as support for reasoning with constraints and preferences. ASP has recently been used as a reliable specification tool in a number of promising applications. For instance, several tasks in information integration, knowledge management, security management, and configuration, which require complex reasoning capabilities, have been successfully tackled using ASP. In particular, these applications have been explored in several recent projects funded by the European Commission (e.g., the projects WASP [61], INFOMIX [35], and ICONS [33]).

Some attractive benefits of ASP are summarized as follows:

- *Full declarativity:* ASP is fully declarative. The order of rules and atoms in a logic program is not important, and, in general, no knowledge of the operational semantics a specific solver adopts is required.
- *Decidability:* ASP programs are, in their basic flavor, decidable. No special restrictions are needed in order to keep this important property.
- *Support of nonmonotonicity:* ASP supports strong negation as well as negation as failure. The latter facilitates default reasoning and nonmonotonic inheritance.
- *Nondeterminism:* Concepts may be defined which "range" over a space of choices without any particular restriction. Combined with extensions for preferences and different kinds of constraints, this enables a compact specification of search and optimization problems.
- *Scalability:* Despite the computational expressiveness of ASP, current state-of-the-art solvers, such as DLV [36], GnT [34], or Cmodels-3 [38], have reached a level of maturity which allows them to deal even with large datasets.

We refer to [60] for a repository of ASP solvers, and to [63] for a comprehensive report on recent ASP applications; a showcase collection is available online at

`http://www.kr.tuwien.ac.at/projects/WASP/showcase.html`.

In the Semantic Web perspective, significant efforts have been made to highlight the benefits of ASP for the Rules Layer of the Semantic Web architecture and its inter-

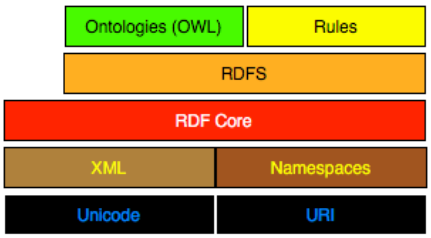

Fig. 1. Ontology and rule languages in the Semantic-Web layer cake

actions with the Ontology Layer. A variety of upcoming applications supports adopting ASP as a formalism for realizing the Rules Layer. The inherent nondeterminism and the possibility to enrich the semantics with weak (i.e., soft) constraints make ASP a well-suited candidate for applications like Web-service matchmaking and ontology alignment [58]. It is worth mentioning that an ASP application for Web-service composition [49] earned first prize in the EEE Web-Service Composition Contest [13].

The remainder of this paper is organized as follows. The next section contains preliminaries on the relevant parts of the Semantic Web architecture, and Section 3 introduces ASP. In Section 4, we point out issues in combining rules and ontologies, and briefly survey approaches in this direction. After that, Section 5 presents nonmonotonic *description-logic programs* (or *dl-programs*, for short) as an example of an approach for combining rules and ontologies. The subsequent Section 6 presents an extension of this approach towards an integration of rules and general external software, in which the usage of higher-order predicates is facilitated. Finally, Section 7 provides a short discussion and concludes the paper.

In order to have a cohesive flow and to illustrate the different ASP extensions, we introduce an example in a storyboard-like fashion, which will serve as a running example throughout the paper.

2 Ontology Formalisms

Rules and ontologies represent two main components in the Semantic-Web vision which are expected to tightly interplay for making this vision a reality. In order to illustrate a plausible scenario where rules and ontologies interact, we will incrementally build a simple, yet conceivable, example.

Example 1 (Motivating Example, Part I). The Reasoning-Web Summer School is planning the organization of its social dinner. In order to make the attendees happy with this event and to make them familiar with ontologies, they decide to ask them to declare their preferences about wines, in terms of a class description reusing the (in)famous Wine Ontology [62]. The organizers realize that only one kind of wine would not achieve the goal of fulfilling all the attendees' preferences. Thus, they aim at automatically finding the cheapest selection of bottles such that any attendee can have his or her preferred wine at the dinner.

The organizers quickly realize that several building blocks are needed to accomplish this task. First of all, a good formalism to express the domain of interest (involving wines, their properties, and bottles) is needed. So they search among the currently available technologies and return with a strange brew of acronyms such as RDF, RDFS, and OWL. □

The realization of reasoning with rules and ontologies affects basically four components of the so-called "Semantic-Web layer cake" [7]: *RDF*, *RDFS*, the *Ontology Layer*, and the *Rules Layer*. A slightly simplified version of this relevant part of the architecture proposal for the Semantic Web is shown in Fig. 1.

Layered on top of standards which mainly serve to provide common syntax for information exchange on the Web, the *Resource Description Framework* (RDF) [57, 27] provides a common flexible data model for the Semantic Web. Based on arbitrary labeled graphs, RDF does not enforce a particular data schema upfront. Next, *RDF Schema* (RDFS) provides facilities to define simple taxonomies among concepts and relations.

While RDFS as such could already be viewed as a simple ontology language, in order to provide more expressiveness for describing formal conceptualizations, the *Ontology Layer* was introduced and is realized by means of the OWL Web Ontology Language [11], which can be seen as a syntactic variant of an expressive description logic.

As we already see in Fig. 1, the "Semantic-Web layer cake" is in fact not strictly layered, since rules and ontologies appear side by side. Whereas RDF, RDFS, and OWL have already achieved an acceptable level of maturity as W3C recommendations, it is not yet completely clear where and how to fit in rules, possibly involving nonmonotonicity, preferences, or other expressive features. Defining a proper standard for integrating the plethora of rules languages around is yet to be investigated by W3C's recently established *Rule Interchange Format* (RIF) working group.[3]

A natural choice of rule languages relevant for the integration of rules and ontologies are those originating from logic programming and nonmonotonic reasoning, in particular languages which are based on the *answer-set programming paradigm* (cf., e.g., [6]), on which we focus here. The latter paradigm is a purely declarative problem-solving formalism which gained increasing momentum in the knowledge-representation community over the last decade.

Before introducing this paradigm in more detail though, we briefly recapitulate the established building blocks RDF(S) and OWL, and discuss their formal underpinnings.

2.1 RDF(S)

The *Resource Description Framework* (RDF) defines the data model for the Semantic Web. Driven by the goal of a least possible commitment to a particular data schema, the simplest possible structure for representing information was chosen in RDF, a labeled graph. An RDF graph can be viewed as a set of its directed edges, commonly represented by triples of form $\langle Subject\ Predicate\ Object \rangle$, also called *statements*. Predicates, also referred to as *properties* in RDF terminology, denote the labels, and link a *resource*, identified by a URI, with another resource, datatype literal, or XML literal.

[3] Cf. `http://www.w3.org/2005/rules/wg/charter`.

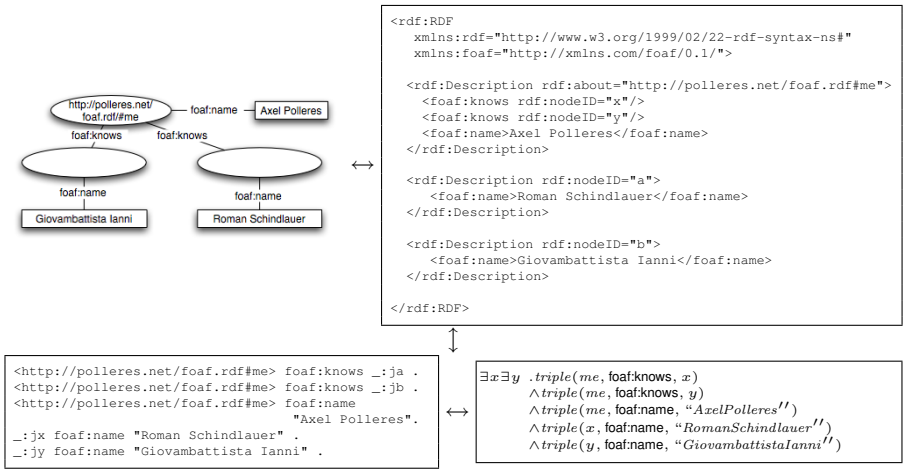

Fig. 2. Different representations of RDF

Moreover, RDF graphs may contain anonymous ("blank") nodes, in order to express incomplete information or queries. Fig. 2 shows an example demonstrating three common notions for RDF graphs: RDF/XML syntax, N-Triples, and representing an RDF graph as a closed first-order formula where blank nodes are conceived as existentially quantified variables. We use the ternary predicate $triple$ to represent RDF statements: alternative representations, like representing triples $\langle S\ P\ O\rangle$ by $P(S, O)$, have some disadvantage for RDF, as we will see below.

This graphs contains the following information: The resource

"`http://polleres.net/foaf.rdf#me`"

with the name "Axel Polleres" knows someone named "Giovambattista Ianni" and someone named "Roman Schindlauer". Terms like foaf:knows are shortcuts for full URIs like `http://xmlns.com/foaf/0.1/knows`,[4] i.e., using so-called *namespace prefixes* from XML, for ease of legibility.

Moreover, basic RDF defines a special property rdf:type, which allows the specification of "is-a" relations, such as, for instance,

$\langle$`http://polleres.net/foaf.rdf#me` rdf:type foaf:Person$\rangle$.

RDF supports two basic types, viz. rdf:Property and rdf:XMLLiteral, and a basic set of XML schema datatypes.

The semantics of RDF can be essentially viewed as corresponding to the first-order representation chosen in Fig. 2 plus entailment of several axiomatic triples, such as that the triple $\langle X$ rdf:type rdf:Property $\rangle$ is an axiom for all X which occur in the predicate

[4] This represents typical information which you might find in a so-called *FOAF description*, an RDF vocabulary for expressing personal information with growing popularity, see `http://www.foaf-project.org/`.

Table 1. Semantics of RDFS

$\forall S, P, O.(triple(S, P, O) \supset triple(P, \mathsf{rdf{:}type}, \mathsf{rdf{:}Property}))$,

$\forall S, P, O.(triple(S, P, O) \supset triple(S, \mathsf{rdf{:}type}, \mathsf{rdfs{:}Resource}))$,

$\forall S, P, O.(triple(S, P, O) \supset triple(O, \mathsf{rdf{:}type}, \mathsf{rdfs{:}Resource}))$,

$\forall S, P, O.(triple(S, P, O), triple(P, \mathsf{rdfs{:}domain}, C)) \supset triple(S, \mathsf{rdf{:}type}, C))$,

$\forall S, P, O, C.(triple(S, P, O) \wedge triple(P, \mathsf{rdfs{:}range}, C) \supset triple(O, \mathsf{rdf{:}type}, C))$,

$\forall C.(triple(C, \mathsf{rdf{:}type}, \mathsf{rdfs{:}Class}) \supset triple(C, \mathsf{rdfs{:}subClassOf}, \mathsf{rdfs{:}Resource})$,

$\forall C_1, C_2, C_3.((triple(C_1, \mathsf{rdfs{:}subClassOf}, C_2) \wedge$
$triple(C_2, \mathsf{rdfs{:}subClassOf}, C_3)) \supset triple(C_1, \mathsf{rdfs{:}subClassOf}, C_3)$,

$\forall S, C_1, C_2.((triple(S, \mathsf{rdf{:}type}, C_1) \wedge$
$triple(C_1, \mathsf{rdfs{:}subClassOf}, C_2)) \supset triple(S, \mathsf{rdf{:}type}, C_2))$,

$\forall S, C.(triple(S, \mathsf{rdf{:}type}, C) \supset triple(C, \mathsf{rdf{:}type}, \mathsf{rdfs{:}Class})$,

$\forall C.(triple(C, \mathsf{rdf{:}type}, \mathsf{rdfs{:}Class}) \supset triple(C, \mathsf{rdfs{:}subClassOf}, C)$,

$\forall P_1, P_2, P_3.((triple(P_1, \mathsf{rdfs{:}subPropertyOf}, P_2) \wedge$
$triple(P_2, \mathsf{rdfs{:}subPropertyOf}, P_3)) \supset triple(P_1, \mathsf{rdfs{:}subPropertyOf}, P_3))$,

$\forall S, P_1, P_2, O.(triple(S, P_1, O) \wedge triple(P_1, \mathsf{rdfs{:}subPropertyOf}, P_2) \supset triple(S, P_2, O))$,

$\forall P.(triple(P, \mathsf{rdf{:}type}, \mathsf{rdf{:}Property}) \supset triple(P, \mathsf{rdfs{:}subPropertyOf}, P))$

position of any other triple. In particular, this also makes, for instance, $\langle$ rdf:type rdf:type rdf:Property $\rangle$ an axiom.

The semantics of RDF involves some more peculiarities in the handling of XML literals, RDF containers, and lists. Most remarkably, it should be noted that the RDF vocabulary contains an infinite number of predefined properties rdf:_1, rdf:_2, ... for container membership, and thus gives rise to an infinite number of axiomatic triples $\langle$rdf:_1 rdf:type rdf:Property$\rangle$, ... We refer the interested reader to [27] for details.

RDF Schema (RDFS) is a semantic extension of basic RDF essentially by giving special meaning to the properties rdfs:subClassOf and rdfs:subPropertyOf, as well as to several types (like rdfs:Class, rdfs:Resource, rdfs:Literal, rdfs:Datatype etc.), in order to express simple taxonomies and hierarchies among properties and resources.

The semantics of RDFS can to a large extent be approximated by a set of sentences of first-order logic (FOL), reusing the notion from above (see Table 1)[5] plus the axiomatic triples from [27, Sections 3.1 and 4.1]. Note that our choice of using a ternary predicate *triple* in favor of a binary representation helped us to avoid higher-order-like rules such as $\forall S, P, O.\ P(S, O) \supset$ rdf:type $(P,$rdf:Property$)$ in this axiomatization. Again, we do not elaborate upon peculiarities and additional rules or axioms in the context of RDF containers and XML literals here.

[5] We use '$\supset$' for material implication to avoid confusion with '$\leftarrow$' as commonly used in logic programming.

Table 2. Expressing OWL DL Property axioms to DL and FOL

OWL property axioms as RDF Triples	DL syntax	FOL short representation
$\langle P$ rdfs:domain $C\rangle$	$\top \sqsubseteq \forall P^-.C$	$\forall x, y : P(x, y) \supset C(x)$
$\langle P$ rdfs:range $C\rangle$	$\top \sqsubseteq \forall P.C$	$\forall x, y : P(x, y) \supset C(y)$
$\langle P$ owl:inverseOf $P_0\rangle$	$P \equiv P_0^-$	$\forall x, y : P(x, y) \equiv P_0(y, x)$
$\langle P$ rdf:type owl:SymmetricProperty $\rangle$	$P \equiv P^-$	$\forall x, y : P(x, y) \equiv P(y, x)$
$\langle P$ rdf:type owl:FunctionalProperty $\rangle$	$\top \sqsubseteq \leqslant 1P$	$\forall x, y_1, y_2 : P(x, y_1) \wedge P(x, y_2) \supset y_1 = y_2$
$\langle P$ rdf:type owl:InverseFunctionalProperty $\rangle$	$\top \sqsubseteq \leqslant 1P^-$	$\forall x_1, x_2, y : P(x_1, y) \wedge P(x_2, y) \supset x_1 = x_2$
$\langle P$ rdf:type owl:TransitiveProperty $\rangle$	$P^+ \sqsubseteq P$	$\forall x, y, z : P(x, y) \wedge P(y, z) \supset P(x, z)$

2.2 Description Logics and the OWL Web Ontology Language

The next layer in the Semantic-Web stack serves to formally define shared conceptualizations, i.e., ontologies [25], on top of the RDF/RDFS data model. In order to formally specify such domain models, the W3C has chosen a language which is close to a syntactic variant of an expressive but still decidable description logic (DL), namely $\mathcal{SHOIN}(D)$. More precisely, the OWL DL variant coincides with this description logic, at the cost of imposing several restrictions on the usage of RDF(S). These restrictions (e.g., disallowing that a resource is used both as a class and an instance) are lifted in OWL Full which combines the description logic flavor of OWL DL and the syntactic freedom of RDF(S). For an in-depth discussion of the peculiarities of OWL Full, we refer the interested reader to the language specification [11] and restrict our observations to OWL DL here.

While RDFS itself may already be viewed as a simple ontology language, OWL adds several features beyond the simple definition of hierarchies (rdfs:subPropertyOf, rdfs:subClassOf) to define relations between properties and classes.

As for properties, OWL allows to specify transitive, symmetric, functional, inverse functional, and inverse properties. The correspondences of respective OWL properties and classes with description logics and first-order logic axioms expressible in OWL can be found in Table 2. Note that we switch to the binary representation $P(S, O)$ of triples here, since in description logics (and thus in OWL DL), predicate names and resources are assumed to be disjoint.

Moreover, OWL allows the specifications of complex class descriptions to be used in rdfs:subClassOf statements. Complex descriptions may involve class definitions in terms of union or intersection of other classes, as well as restrictions on properties. Table 3 gives an overview of the expressive possibilities of OWL for class descriptions and its semantic correspondences with description logics and first-order logics.[6] Such class descriptions can be related to each other using rdfs:subClassOf, owl:equivalentClass, or owl:disjointWith keywords, which allow us to express description-logic axioms of the form $C_1 \sqsubseteq C_2$, $C_1 \equiv C_2$, or $C_1 \sqcap C_2 \sqsubseteq \bot$, respectively, in OWL.

Finally, OWL allows to express explicit equality or inequality relations between individuals by means of the owl:sameAs and owl:differentFrom properties, e.g., the triples

[6] We use a simplified notion for the first-order logic translation here—actually, the translation needs to be applied recursively for any complex description-logic term. For a formal specification of the correspondence between description-logic expressions and first-order logic, cf. [5].

Table 3. Mapping of OWL DL Complex Class Descriptions to DL and FOL

OWL complex class descriptions*	**DL syntax**	**FOL short representation**
owl:Thing	$\top$	$x = x$
owl:Nothing	$\bot$	$\neg x = x$
owl:intersectionOf $(C_1 \dots C_n)$	$C_1 \sqcap \dots \sqcap C_n$	$\bigwedge C_i(x)$
owl:unionOf $(C_1 \dots C_n)$	$C_1 \sqcup \dots \sqcup C_n$	$\bigvee C_i(x)$
owl:complementOf (C)	$\neg C$	$\neg C(x)$
owl:oneOf $(o_1 \dots o_n)$	$\{o_1 \dots o_n\}$	$\bigvee x = o_i$
owl:restriction (P owl:someValuesFrom (C))	$\exists P.C$	$\exists y.P(x,y) \wedge C(y)$
owl:restriction (P owl:allValuesFrom (C))	$\forall P.C$	$\forall y.P(x,y) \supset C(y)$
owl:restriction (P owl:value (o))	$\exists P.\{o\}$	$P(x,o)$
owl:restriction (P owl:minCardinality (n))	$\geqslant nP$	$\exists^n_{i=1} y_i.\, \bigwedge^n_{j=1} P(x,y_j) \wedge \bigwedge_{i \neq j} y_i \neq y_j$
owl:restriction (P owl:maxCardinality (n))	$\leqslant nP$	$\forall^{n+1}_{i=1} y_i.(\bigwedge^n_{j=1} P(x,y_i) \supset \bigvee_{i \neq j} y_i = y_j)$

*For reasons of legibility, we use a variant of the OWL abstract syntax [47] in this table.

$$\langle \texttt{http://www.polleres.net/foaf.rdf\#me}\ \text{owl:sameAs}\ \texttt{http://polleres.net/foaf.rdf\#me} \rangle$$

and

$$\langle \texttt{http://polleres.net/foaf.rdf\#me}\ \text{owl:differentFrom}\ \texttt{http://www.gibbi.com/foaf.rdf\#me} \rangle$$

boil down to

$$\texttt{http://www.polleres.net/foaf.rdf\#me} = \texttt{http://polleres.net/foaf.rdf\#me}$$
$$\wedge\, \texttt{http://polleres.net/foaf.rdf\#me} \neq \texttt{http://www.gibbi.com/foaf.rdf\#me}.$$

For details on the description logics notion used in the Tables 2 and 3, we refer the interested reader to, e.g., [5]. For our purposes, basic understanding of the corresponding definitions in term of first-order logic will be sufficient. What makes description logics the formalism of choice is the fact that it defines a *decidable fragment* of first-order logic, i.e., queries for entailment of subclass relationships or class membership of a particular individual are effectively computable.

Example 2 (Ontologies in Description Logics). Taking the wine ontology from [62], let us illustrate some of the conceptualizations therein in their corresponding description-logics syntax:

$$\begin{aligned} Wine &\sqsubseteq PotableLiquid \sqcap = 1 hasMaker \sqcap \forall hasMaker.Winery; \\ Wine &\sqsubseteq \geq 1 madeFromGrape \sqcap = 1 hasFlavor; \\ \forall hasColor^{-}.T &\sqsubseteq \{\text{“}White\text{”}, \text{“}Rose\text{”}, \text{“}Red\text{”}\}; \\ WhiteWine &\equiv Wine \sqcap \forall hasColor.\{\text{“}White\text{”}\}. \end{aligned}$$

This knowledge base expresses the following information: A wine is a potable liquid, having exactly one maker, who is a member of the class $Winery$. Moreover, wines are

made from at least one sort of grapes and have exactly one of the flavors, and one of the colors "$White$", "$Rose$", and "Red". A $WhiteWine$ is a wine with color "$White$". Finally, $Welschriesling$ is an instance of $WhiteWine$. □

3 Answer-Set Programming

After having introduced some foundations of the Semantic Web in terms of a data model (RDF) and ontology languages (RDFS and OWL), let us now turn to logic programs as a way to realize the Semantic-Web Rules Layer. For illustration purposes, consider the following continuation of our running example:

Example 3 (Motivating Example, Part II). As soon as the wine domain is described, the social-dinner organizers now have to face the problem of quickly modeling rules that describe a set of bottles that are suitable for all the participants, and to express the choice criteria among these candidate sets. They realize soon that domain-description languages accomplished their job well, but now they need some different tool: First, how to express possible choices of bottles? How to determine the set of attendees (say, the class $nonSatisfied$) that are *not* assigned a compliant bottle? Unfortunately, under an open-world assumption, no attendee can be entailed as belonging to this class. Moreover, is it possible to exclude the situations where $nonSatisfied$ is non-empty, and where the price of this selection of bottles is possibly minimal?

They conclude that a rule-based formalism with disjunction and nonmonotonic features would be the most appropriate formalism, and, among others, choose to investigate on the characteristics of ASP (answer-set programming). □

Answer-set programming has its roots in the seminal work by Gelfond and Lifschitz [22], who presented a semantics for logic programs with negation as failure and strong negation, where multiple *answer sets* (or *stable models*) may be ascribed to a program. This inherent nondeterminism can be exploited to represent different solutions to a problem in the answer sets of a logic program, as fostered, e.g., in [39, 42, 44].

3.1 Syntax

Let Φ be a first-order vocabulary with nonempty finite sets of constant and predicate symbols, but no function symbols.[7] Let $\mathcal{X}$ be a set of variables. A *term* is either a variable from $\mathcal{X}$ or a constant symbol from Φ. An *atom* is an expression of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol of arity $n \geq 0$ from Φ, and $t_1, \ldots, t_n$ are terms. A *literal* l is either an atom or an expression of form $-p$, where "$-$" denotes *strong negation* and p is an atom. The *complementary literal* $-l$ of l is $-p$ if $l = p$ and p if $l = -p$. A *negation-as-failure literal* (or *NAF-literal*) is either a literal or an

[7] Gelfond and Lifschitz allowed function symbols and inconsistent answer sets in their seminal paper [22]. Current ASP solvers have limited support of function symbols, while inconsistent answer sets are not allowed as valid answers.

expression of form $not\ l$, where "not" denotes *negation as failure*, or *default negation*, and l is a literal. A *disjunctive rule* (or simply a *rule*) r is an expression of the form

$$a_1 \vee \cdots \vee a_l \leftarrow b_1, \ldots, b_k, not\, b_{k+1}, \ldots, not\, b_m, \tag{1}$$

where $l \geq 0$, $m \geq k \geq 0$, and all a_i and b_j are literals. The disjunction $a_1 \vee \cdots \vee a_l$ is the *head* of r, while the conjunction $b_1, \ldots, b_k, not\, b_{k+1}, \ldots, not\, b_m$ is the *body* of r, where $b_1, \ldots, b_k$ (resp., $not\, b_{k+1}, \ldots, not\, b_m$) is the *positive* (resp., *negative*) *body* of r. We use $H(r)$ to denote the set of all head literals $\{a_1, \ldots, a_l\}$ of r, and $B(r)$ to denote the set of all body literals $B^+(r) \cup B^-(r)$ of r, where $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$.

A *disjunctive program* (or simply *program*) P is a finite set of (disjunctive) rules.

If the body of a rule r is empty (i.e., if $B(r) = \emptyset$), then r is a *fact*, and we often omit "$\leftarrow$" in such a case. A rule is *positive* if $B^-(r) = \emptyset$, and *normal* if the head of r is a literal. Similarly, a program is *positive* resp. *normal*, if each rule in it is positive resp. normal. A rule without head literals is an (*integrity*) *constraint*.

Example 4 (Simple Wine Program). The following program is a simplistic representation of a part of the wine scenario described previously, in which a plain ontology is natively represented within the logic program.

% A suite of wine bottles and their kinds

$wineBottle(\text{“}SelaksIceWine\text{”});$ $isA(\text{“}SelaksIceWine\text{”}, \text{“}whiteWine\text{”});$
$isA(\text{“}SelaksIceWine\text{”}, \text{“}sweetWine\text{”});$

$wineBottle(\text{“}CheninBlanc\text{”});$ $isA(\text{“}CheninBlanc\text{”}, \text{“}whiteWine\text{”});$
$isA(\text{“}CheninBlanc\text{”}, \text{“}dryWine\text{”});$

$wineBottle(\text{“}Chardonnay\text{”});$ $isA(\text{“}Chardonnay\text{”}, \text{“}whiteWine\text{”});$
$isA(\text{“}Chardonnay\text{”}, \text{“}dryWine\text{”});$

$wineBottle(\text{“}ChiantiClassico\text{”});$ $isA(\text{“}ChiantiClassico\text{”}, \text{“}redWine\text{”});$
$isA(\text{“}ChiantiClassico\text{”}, \text{“}dryWine\text{”});$

$wineBottle(\text{“}TaylorPort\text{”});$ $isA(\text{“}TaylorPort\text{”}, \text{“}redWine\text{”});$
$isA(\text{“}TaylorPort\text{”}, \text{“}sweetWine\text{”}).$

% Persons and their preferences

$person(\text{“}axel\text{”});$ $preferredWine(\text{“}axel\text{”}, \text{“}whiteWine\text{”});$
$person(\text{“}gibbi\text{”});$ $preferredWine(\text{“}gibbi\text{”}, \text{“}redWine\text{”});$
$person(\text{“}roman\text{”});$ $preferredWine(\text{“}roman\text{”}, \text{“}dryWine\text{”}).$

% Available bottles a person likes

$compliantBottle(X, Z) \leftarrow preferredWine(X, Y), isA(Z, Y).$

The last rule describes bottles which are compliant with a person's preference. □

Let us now consider a more elaborate version of this program.

Example 5 (Wine Program II). Compared to Example 4, we add the following rules:

$doesNotLike(X, Z) \leftarrow person(X), wineBottle(Z), not\ compliantBottle(X, Z).$

% This rule generates multiple answer sets

$bottleChosen(X) \vee -bottleChosen(X) \leftarrow compliantBottle(Y, X).$

% Ensure that each person gets a bottle
$hasBottleChosen(X) \leftarrow bottleChosen(Z), compliantBottle(X, Z);$
$\leftarrow person(X), not\ hasBottleChosen(X).$

The first rule concludes that somebody does not like wine bottles which do no comply with the personal desires. The second rule generates different worlds: ones in which a given bottle is chosen and others in which it is not. The third rule, together with the constraint, prunes all worlds (under closed-world assumption, CWA) in which some person has no bottle chosen.

Moreover, note that the second rule (the "choice" rule) may be equivalently replaced with

$-bottleChosen(X) \leftarrow not\ bottleChosen(X), compliantBottle(Y, X);$
$bottleChosen(X) \leftarrow not\ -bottleChosen(X), compliantBottle(Y, X).$

Under the answer-set semantics (introduced next), this pair of rules enforces that either $bottleChosen(X)$ or $-bottleChosen(X)$ is included in an answer set (but not both), providing it contains $compliantBottle(Y, X)$. □

3.2 Semantics

The *Herbrand universe* of a program P, denoted HU_P, is the set of all constant symbols appearing in P. If there is no such constant symbol, then $HU_P = \{c\}$, where c is an arbitrary constant symbol from Φ. As usual, terms, atoms, literals, rules, programs, etc. are *ground* iff they do not contain any variables. The *Herbrand base* of a program P, denoted HB_P, is the set of all ground (classical) literals that can be constructed from the predicate symbols appearing in P and the constant symbols in HU_P. A *ground instance* of a rule $r \in P$ is obtained from r by replacing every variable that occurs in r by a constant symbol from HU_P. We use $ground(P)$ to denote the set of all ground instances of rules in P.

A set of literals $X \subseteq HB_P$ is *consistent* iff $\{p, -p\} \not\subseteq X$ for every atom $p \in HB_P$. An *interpretation* I relative to a program P is a consistent subset of HB_P. A *model* of a positive program P is an interpretation $I \subseteq HB_P$ such that $B(r) \subseteq I$ implies $H(r) \cap I \neq \emptyset$, for every $r \in ground(P)$. An *answer set* of a positive program P is a minimal model of P with respect to set inclusion. In particular, if P is positive and does not involve disjunction, then there exists a single answer set (if one exists).

Example 6 (Simple Wine Program, continued). Our simple wine program does not contain disjunction. Its Herbrand universe is

HU_P={"$SelaksIceWine$", "$CheninBlanc$", "$Chardonnay$", "$ChiantiClassico$",
"$TaylorPort$", "$whiteWine$", "$redWine$", "$sweetWine$", "$dryWine$",
"$axel$", "$gibbi$", "$roman$"}

and its single answer set consists of all the facts of the program, together with the following items:

$compliantBottle(\text{“}axel\text{”}, \text{“}SelaksIceWine\text{”});$
$compliantBottle(\text{“}axel\text{”}, \text{“}CheninBlanc\text{”});$
$compliantBottle(\text{“}axel\text{”}, \text{“}Chardonnay\text{”});$
$compliantBottle(\text{“}gibbi\text{”}, \text{“}ChiantiClassico\text{”});$
$compliantBottle(\text{“}gibbi\text{”}, \text{“}TaylorPort\text{”});$
$compliantBottle(\text{“}roman\text{”}, \text{“}CheninBlanc\text{”});$
$compliantBottle(\text{“}roman\text{”}, \text{“}Chardonnay\text{”});$
$compliantBottle(\text{“}roman\text{”}, \text{“}ChiantiClassico\text{”}).$ □

The *Gelfond-Lifschitz reduct* [22] of a program P relative to an interpretation $I \subseteq HB_P$, denoted P^I, is the ground positive program that is obtained from $ground(P)$ by

(i) deleting every rule r such that $B^-(r) \cap I \neq \emptyset$, and
(ii) deleting the negative body from every remaining rule.

An *answer set* of a program P is an interpretation $I \subseteq HB_P$ such that I is an answer set of P^I.

Note that, for positive P, $P^I = ground(P)$, and thus the answer sets of P are its minimal models, as we recall from above. This applies to the program in Example 4.

Example 7 (Wine Program II, continued). Let us extend the answer set of the program in Example 4 by the atoms

$doesNotLike(\text{“}axel\text{”}, \text{“}ChiantiClassico\text{”}), doesNotLike(\text{“}axel\text{”}, \text{“}TaylorPort\text{”}),$
$doesNotLike(\text{“}gibbi\text{”}, \text{“}SelaksIceWine\text{”}), doesNotLike(\text{“}gibbi\text{”}, \text{“}CheninBlanc\text{”}),$
$doesNotLike(\text{“}gibbi\text{”}, \text{“}Chardonnay\text{”}), doesNotLike(\text{“}roman\text{”}, \text{“}SelaksIceWine\text{”}),$
$doesNotLike(\text{“}roman\text{”}, \text{“}TaylorPort\text{”}), -bottleChosen(\text{“}SelaksIceWine\text{”}),$
$-bottleChosen(\text{“}CheninBlanc\text{”}), bottleChosen(\text{“}Chardonnay\text{”}),$
$bottleChosen(\text{“}ChiantiClassico\text{”}), -bottleChosen(\text{“}TaylorPort\text{”}),$
$hasBottleChosen(\text{“}axel\text{”}), hasBottleChosen(\text{“}roman\text{”}),$
$hasBottleChosen(\text{“}gibbi\text{”}),$

and let I be the resulting interpretation. Then, the program P^I contains all ground instances of positive rules on HU_P, plus the rules (originally containing negation in P)

$$doesNotLike(c, c') \leftarrow person(c), wineBottle(c'),$$

where (c, c') is from the set

$\{(\text{“}axel\text{”}, \text{“}ChiantiClassico\text{”}), (\text{“}axel\text{”}, \text{“}TaylorPort\text{”}), (\text{“}gibbi\text{”}, \text{“}SelaksIceWine\text{”}),$
$(\text{“}gibbi\text{”}, \text{“}CheninBlanc\text{”}), (\text{“}gibbi\text{”}, \text{“}Chardonnay\text{”}), (\text{“}roman\text{”}, \text{“}TaylorPort\text{”}),$
$(\text{“}roman\text{”}, \text{“}SelaksIceWine\text{”})\}.$

As easily checked, I satisfies all rules in P^I, and moreover is a minimal model of P^I. Therefore, I is an answer set of P. However, other answer sets do exist. □

3.3 Reasoning Tasks

The main reasoning tasks associated with programs under the answer-set semantics are the following:

- decide whether a given program P has an answer set;
- given a program P and ground literals $l_1, \ldots, l_n$, decide whether $l_1, \ldots, l_n$ simultaneously hold in every (resp., some) answer set of P (this is known as *cautious* resp. *brave reasoning*);
- given a program P and nonground literals $l_1, \ldots, l_n$ over variables $X_1, \ldots, X_k$, list all assignments ν of values to $X_1, \ldots, X_k$ such that $l_1\nu, \ldots, l_n\nu$ is cautiously (resp., bravely) true (*query answering*); and
- compute the set of all answer sets of a given program P.

Example 8 (Simple Wine Program, continued). In our simple wine program, we have a single answer set, and thus cautious and brave reasoning coincides. For instance, $compliantBottle(\text{“}axel\text{”}, \text{“}SelaksIceWine\text{”})$ is both a cautious as well as a brave consequence of the program. For the query $person(X)$, we obtain the answers “$axel$”, “$gibbi$”, and “$roman$”. □

Example 9 (Wine Program II, continued). The more elaborated wine program has 20 answer sets, corresponding to the possibilities whether a bottle is being chosen or not. The cautious query $bottleChosen(\text{“}SelaksIceWine\text{”})$ fails, while the brave query $bottleChosen(\text{“}SelaksIceWine\text{”})$ succeeds. For the query $bottleChosen(X)$, we obtain no answer under cautious reasoning. □

The basic ASP language, as introduced above, has been extended in the literature with many features like *weak constraints* [8], *aggregates* [20] (as familiar from database query languages), or *cardinality* and *weight constraints* [45]. The fruitful combination of these features allowed ASP to become an important knowledge-representation formalism for declaratively solving AI problems.

Example 10 (Wine Program III). Suppose we want to single out situations in which a smallest number of bottles is chosen. This is effected in DLV [36] by the weak constraint

$$:\sim bottleChosen(X)\ [1].$$

Intuitively, each fact $bottleChosen(c)$ in an answer set is assigned a penalty of 1, and total penalties are minimized. In our example, the optimum are two bottles (e.g., $bottleChosen(\text{“}Chardonnay\text{”})$ and $bottleChosen(\text{“}ChiantiClassico\text{”})$). For a formal definition of the syntax and semantics of weak constraints, and a refinement using priority levels, we refer to [36]. □

4 Combining Rules with Ontologies

Motivated by our wine selection example, we have illustrated that answer-set programming might be a good candidate for filling the gap extending the Semantic-Web layers

with a suitable rules component. However, there are several obstacles in finding the right combination of rich ontology languages such as OWL, which are based on classical logic, with logic-programming based languages such as answer-set programming (see also [53] for a discussion).

4.1 Logic Programming vs. Classical Logic

As well-known, the core of logic programming, i.e., definite positive programs, has a direct correspondence with the Horn subset of classical first-order logic. To wit, a rule of form (1) which is definite (i.e., when $l = 1$) and not-free (i.e., when $m = k$) can be read as a first-order sentence

$$(\forall)\, b_1 \wedge \ldots \wedge b_k \supset a \tag{2}$$

where $(\forall)$ denotes the universal closure operator. This subset of first-order logic allows for a sound and complete decision procedure for entailment of ground atomic formulae, which is in the function-free (datalog) case computable in finite polynomial time.

However, there are some slight but important differences between the logic-programming view and the first-order view already for definite programs.

Non-ground entailment. The first divergence becomes apparent already in case of positive programs. The logic-programming semantics is defined in terms of minimal Herbrand models, i.e., sets of ground facts. Take for example the logic program

$potableLiquid(X) \leftarrow wine(X)$;
$wine(X) \leftarrow whiteWine(X)$;
$whiteWine(\text{“}WelschRiesling\text{”})$.

Both the logic-program reading and the Horn-clause reading of this program yields the entailment of facts $whiteWine(\text{“}WelschRiesling\text{”})$, $wine(\text{“}WelschRiesling\text{”})$, and $potableLiquid(\text{“}WelschRiesling\text{”})$. The first-order reading of the program would allow further non-factual inferences, such as

$wine(\text{“}WelschRiesling\text{”}) \supset potableLiquid(\text{“}WelschRiesling\text{”})$ and
$\forall\, X\, .whiteWine(X) \supset PotableLiquid(X)$,

which are not entailed by the logic program. Logic programs, minimal Herbrand models (and answer sets as their extension) are mainly concerned with facts.

Negation as failure vs. classical negation. Divergences become more severe when considering programs with negation. Negation as failure not is evaluated with respect to a closed-world assumption (CWA) whereas negation in description logics and thus in OWL (owl:complementOf) is interpreted classically. Let us again demonstrate this with a small example:

$wine(X) \leftarrow whiteWine(X)$;
$nonWhite(X) \leftarrow not\ whiteWine(X)$;
$wine(myDrink)$.

Not given any additional information, under the answer-set semantics this program entails both bravely and cautiously the fact $nonWhite(myDrink)$. However, this conclusion would not be justified in a first-order or description-logics reading of the above program, such as:

$$\begin{array}{ll} \forall X.\,(WhiteWine(X) \supset Wine(X)) \wedge & WhiteWine \sqsubseteq Wine \\ \forall X.\,(\neg WhiteWine(X) \supset NonWhite(X)) \wedge & \neg WhiteWine \sqsubseteq NonWhite \\ Wine(myDrink). & myDrink \in Wine. \end{array}$$

The reason for this is the different purposes classical negation and negation as failure serve: the latter to be understood as modeling (defeasible) default assumptions with nonmonotonic behavior. While some people argue that such a kind of nonmonotonic negation is unsuitable for an open environment like the Web, there are several applications, e.g., in information integration, where negation as failure has proved particularly useful (see Subsection 5.3).

Strong negation vs. classical negation. Note that also strong negation, as used in ASP has a slightly different flavor than its classical counterpart. That is, the following two representations of a logic program and an OWL knowledge base again slightly diverge:

$$\begin{array}{ll} Wine(X) \leftarrow Whitewine(X); & Whitewine \sqsubseteq Wine; \\ -Wine(myDrink). & myDrink \in \neg Wine. \end{array}$$

Whereas the description-logic knowledge base would entail $myDrink \in \neg whiteWine$, the corresponding fact $-whiteWine(myDrink)$ is not a justified conclusion in a logic-programming setting, i.e., neither the law of the excluded middle nor contraposition does hold upfront in ASP. Nonetheless, one can "emulate" classical behavior of certain predicates in ASP. For instance, adding a rule $whiteWine(X) \vee -whiteWine(X)$ in the above example would achieve this.

Logic Programming and equality. Answer-set programming engines typically deploy a unique-names assumption (UNA) and do not support real equality reasoning, i.e., equality in the head of rules. This does not comply necessarily with the view in classical logic, and thus with RDF and OWL, where no such assumption is made. While equality "$=$" and inequality "$\neq$" predicates are allowed in rule bodies, they represent syntactic equality and (default) negation thereof only. This shall not be confused with OWL's owl:sameAs and owl:differentFrom directives. Following up the example from Section 2.2, consider the following rule base:

$knowsOtherPeople(X) \leftarrow knows(X, Y), X \neq Y;$
$knows($"`http://polleres.net/foaf.rdf\#me`",
"`http://www.polleres.net/foaf.rdf\#me`").

Under standard ASP semantics where UNA is deployed, "$\neq$" amounts to "$not\ =$". Thus,

$knowsOtherPeople($"`http://polleres.net/foaf.rdf\#me`")

would be entailed.

Enabling reasoning with equality has usually a very high computational cost. Indeed, common description-logic reasoners like FACT++ [55] or RACER [26] also do not support full equality reasoning and nominals.

Decidability. Finally, the probably largest obstacle towards combining the description-logics world of OWL and the logic-programming world of ASP stems from the fact that these two worlds face undecidability issues from two completely different angles.

Indeed, decidability of ASP follows from the fact that it is based on function-free Horn logic where ground entailment can be determined by checking finite subsets of the Herbrand base, i.e., decidability and termination of evaluation strategies is guaranteed by the finiteness of the domain. However, this is not so for description logics. Decidability of reasoning tasks such as satisfiability, class subsumption, or class membership in description logics is often strictly dependent from the combination of constructs which are allowed in the terminological language.

It is often possible to prove decidability by means of the so called *tree-model property*. This property basically says that a description-logic knowledge base has a model iff it has a finite tree shaped model whose depth and branching factor are bounded by the size of the knowledge base [5]. In general, it is possible to attempt to prove decidability by means of a generic finite-model property, although it is worth noting that $\mathcal{SHOIN}$ neither has the tree-model property nor the finite-model property [32].

Unfortunately, it is difficult to combine two decidable fragments coming from the two worlds. As shown in [37], the naive combination of even a very simple description logic with an arbitrary Horn logic is undecidable.

4.2 Strategies for Combining Rules and Ontologies

As one can expect by the above-mentioned problems, combining the two worlds of logic programming and classical logic, underlying description logics, is not straightforward.

However, a naive combination of description logics and Horn rules could be imagined as a possible approach for the Rules Layer of the Semantic Web. Indeed, the *Semantic Web Rule Language* (SWRL) [31] proposal, a recent W3C member submission, straightforwardly extends OWL DL in this spirit. Given an OWL knowledge base, SWRL allows to extend it by Horn rules using unary and binary atoms representing classes (concepts) and roles (properties), respectively. This allows, for instance, combined knowledge bases such as the following:

$$\begin{array}{l} shareFood(W1, W2) \leftarrow hasDrink(D, W1), hasDrink(D, W2), \\ Whitewine \sqsubseteq Wine; \\ \text{“}Trout\ grilled\text{”} \in Dish; \\ (\text{“}Trout\ grilled\text{”}, \text{“}WelschRiesling\text{”}) \in hasDrink, \end{array} \tag{3}$$

where the definition of the role “$shareFood$” by means of the first rule is not expressible directly in description logic alone. However, as mentioned above, this freedom comes at he cost of undecidability in the general case.

On the other extreme, the overcautious approach of allowing interoperability only on the intersection of description logics and Horn logic seems to be too restricted.

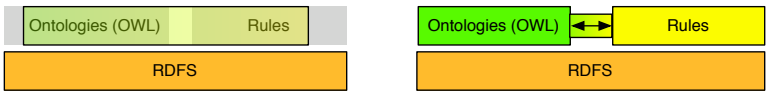

Fig. 3. Integrating Ontologies and Rules by defining "safe interaction" (left) vs. "safe interfaces" (right)

Grosof *et al*. [24] have defined this intersection where the logic-programming and description-logic worlds coincides which they call DLP. However, such an approach leaves a rule and ontology language with very restrictive expressivity. Layering several extensions in the direction of logic programming and ASP on top of the DLP fragment have lead to the *Web Rule Language* (WRL) [2] proposal, an alternative W3C member submission.

In the following, we want to take a closer look at approaches which go beyond DLP but still retain decidability in a more cautious integration than SWRL. Especially, when we want to combine full description logics with full answer-set programming (i.e., not only Horn Rules), things become more involved. In principle, the different approaches in the literature can be divided into two major streams, as described below.

Interaction of ontologies and rules with tight semantic integration. Rules are introduced by adapting existing semantics for rule languages directly in the Ontology Layer. The DLP fragment on the one end and the undecidable SWRL approach on the other mark two extremes of this stream. Nonetheless, in between, recently several proposals have been made to extend expressiveness while still retaining decidability, remarkably several attempts in the ASP field. Common to these approaches are syntactic restrictions of the combined language in a way that guarantees "safe interaction" of the rules and ontologies parts of the language (see Fig. 3).

The first such approach, $\mathcal{AL}$-Log [12], extends the description logic $\mathcal{AL}$ by Horn rules, but with the additional "safety" restriction that every variable of a rule r must appear in at least one of the rule atoms occurring in the body of r, where rule atoms are those predicates which do not appear in the description-logic knowledge base part, but only in rules. This restriction, which retains decidability, is for instance violated by (3). The decidability result for such so-called *DL-safe rules* is extended to a more expressive description logic $\mathcal{SHIQ}$ in [43] bringing us closer to OWL.

Another approach [29] in this direction shows decidability for query answering in $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ with DL-safe rules by an embedding in extended conceptual logic programming, a decidable extension of the answer-set semantics by open domains.

The most recent work in this direction [51–53] loosens the safety restriction further, by allowing non-rule atoms also in rule heads, and also gives a nonmonotonic semantics for non-Horn rules in the spirit of answer-set programming.

Integration of ontologies and rules with strict semantic separation. In this setting, ASP should play a central role in the Rules Layer, while OWL/RDF flavors would keep their purpose of description languages, not aimed at intensive reasoning jobs, in the underlying Ontology Layer. The two layers are kept strictly separate and only communicate

via a "safe interface", but do not impose syntactic restrictions on either the rules or the ontology part (see again Fig. 3).

From the Rules Layer point of view, ontologies are dealt with as an external source of information whose semantics is treated separately. Nonmonotonic reasoning and rules are allowed in a decidable setting, as well as arbitrary mixing of closed and open world reasoning. This approach typically involves special predicates in rule bodies which allow queries to a description-logic knowledge base, and exchange factual knowledge, Examples for this type of interaction are [18, 14, 41] and the call of external description-logic reasoners in the TRIPLE [54] rules engine. In the remainder of this paper, we will focus on nonmonotonic *description-logic programs* [18, 14] as a showcase solution among these approaches.

For excellent surveys which classify the numerous proposals for combining rules and ontologies we refer the interested reader to [4, 46].

5 Nonmonotonic Description-Logic Programs

In this section, we introduce *description-logic programs* (or simply *dl-programs*), which are a novel combination of normal programs and description-logic knowledge bases.

5.1 Syntax

Informally, a dl-program consists of a description-logic knowledge base L and a generalized program P, which may contain queries to L. Roughly, in such a query, it is asked whether a certain description-logic axiom or its negation logically follows from L or not.

A *dl-query* $Q(\mathbf{t})$ is either

(a) a concept inclusion axiom F or its negation $\neg F$; or
(b) of the form $C(t)$ or $\neg C(t)$, where C is a concept and t is a term; or
(c) of the form $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, where R is a role and t_1, t_2 are terms.

A *dl-atom* is an expression of the form

$$DL[S_1 op_1 p_1, \ldots, S_m op_m\ p_m; Q](\mathbf{t})\,, \tag{4}$$

where $m \geq 0$, and such that each S_i is either a concept or a role, $op_i \in \{\uplus, \sqcup\}$, p_i is a unary resp. binary predicate symbol, and $Q(\mathbf{t})$ is a dl-query. We call $p_1, \ldots, p_m$ the *input predicate symbols* of (4). Intuitively, $op_i = \uplus$ (resp., $op_i = \sqcup$) increases S_i (resp., $\neg S_i$) by the extension of p_i.

A *dl-rule* r has the form (1),[8] where any literal $b_1, \ldots, b_m \in B(r)$ may be a dl-atom. We denote by $\widetilde{B}^+(r)$ (resp., $\widetilde{B}^-(r)$) the set of all dl-atoms in $B^+(r)$ (resp., $B^-(r)$). A *dl-program* $KB = (L, P)$ consists of a description-logic knowledge base L and a finite set P of dl-rules.

Positive and normal dl-rules are defined like for ordinary programs. A dl-program $KB = (L, P)$ is *positive*, if P is "not"-free, and is *normal*, if rule heads are literals (i.e., if $l = 1$ in (1)).

We illustrate dl-programs in terms of our running example.

[8] In [18], only rules with $l = 1$ are considered; the extension to arbitrary l is straightforward.

Example 11 (Wine program, OWL). Suppose now that an ontology is available, formulated in OWL, which describes information about available wine bottles (as instances of a concept $Wine$), and contains (among others) further concepts $SweetWine$, $DryWine$, $RedWine$, and $WhiteWine$ for different types of wine. The earlier program is modified by fetching the wines now from the ontology, using the following rule:

% A suite of wine bottles and their kinds
$wineBottle(X) \leftarrow DL[\text{“}Wine\text{”}](X).$

The isA predicate can then be defined by means of the following rules:

% A suite of wine bottles and their kinds
$isA(X, \text{“}sweetWine\text{”}) \leftarrow wineBottle(X), DL[\text{“}SweetWine\text{”}](X);$
$isA(X, \text{“}dryWine\text{”}) \leftarrow wineBottle(X), DL[\text{“}dryWine\text{”}](X);$
$isA(X, \text{“}redWine\text{”}) \leftarrow wineBottle(X), DL[\text{“}redWine\text{”}](X);$
$isA(X, \text{“}whiteWine\text{”}) \leftarrow wineBottle(X), DL[\text{“}WhiteWine\text{”}](X).$

However, the *isA* predicate may be eliminated; instead of

$$compliantBottle(X, Z) \leftarrow preferredWine(X, Y), isA(Z, Y),$$

we may write simply use

$$compliantBottle(X, Z) \leftarrow preferredWine(X, c), wineBottle(Z), DL[c](Z),$$

for each $c \in \{\text{“}SweetWine\text{”}, \text{“}DryWine\text{”}, \text{“}RedWine\text{”}, \text{“}WhiteWine\text{”}\}$. The resulting program is depicted in Fig. 4. Notice that Rules (5)–(12) form a positive normal dl-program. □

Example 12 (Wine program, OWL II). Suppose now that we learn that there is a bottle, “$SelaksIceWine$”, which is a white wine and not dry. We may add this information to the logic program using the facts

$$white(\text{“}SelaksIceWine\text{”}) \text{ and } not\_dry(\text{“}SelaksIceWine\text{”}).$$

In our program, we may pass this information to the ontology by adding in the dl-atoms the operations

$$\text{“}WhiteWine\text{”} \uplus white \text{ and } \text{“}DryWine\text{”} \uplus not\_dry.$$

For instance, $DL[\text{“}Wine\text{”}](X)$ is changed to $DL[\text{“}WhiteWine\text{”} \uplus white, \text{“}DryWine\text{”} \uplus not\_dry; \text{“}Wine\text{”}](X)$. □

5.2 Semantics

We first define Herbrand interpretations and the truth of dl-programs in Herbrand interpretations. In the sequel, let $KB = (L, P)$ be a dl-program.

The *Herbrand base* of P, denoted HB_P, is the set of all ground literals with a standard predicate symbol that occurs in P and constant symbols in Φ. We denote by DL_P be the set of all ground instances of dl-atoms with constant symbols in Φ.

An *interpretation I relative to P* is a consistent subset of HB_P. We say that I is a *model* of $\ell \in HB_P$ under L, denoted $I \models_L \ell$, iff $\ell \in I$, and of a ground dl-atom a of form (4) under L, denoted $I \models_L a$, iff $L \cup \bigcup_{i=1}^{m} A_i(I) \models Q(\mathbf{t})$, where

% A suite of wine bottles and their kinds:

$$wineBottle(X) \leftarrow DL[\text{“}Wine\text{”}](X). \tag{5}$$

% Persons and their preferences:

$$person(\text{“}axel\text{”}); \quad preferredWine(\text{“}axel\text{”}, \text{“}whiteWine\text{”}); \tag{6}$$

$$person(\text{“}gibbi\text{”}); \quad preferredWine(\text{“}gibbi\text{”}, \text{“}redWine\text{”}); \tag{7}$$

$$person(\text{“}roman\text{”}); \quad preferredWine(\text{“}roman\text{”}, \text{“}dryWine\text{”}). \tag{8}$$

% Available bottles a person likes:

$$compliantBottle(X, Z) \leftarrow preferredWine(X, \text{“}SweetWine\text{”}), wineBottle(Z), DL[\text{“}SweetWine\text{”}](Z); \tag{9}$$

$$compliantBottle(X, Z) \leftarrow preferredWine(X, \text{“}DryWine\text{”}), wineBottle(Z), DL[\text{“}DryWine\text{”}](Z); \tag{10}$$

$$compliantBottle(X, Z) \leftarrow preferredWine(X, \text{“}RedWine\text{”}), wineBottle(Z), DL[\text{“}RedWine\text{”}](Z); \tag{11}$$

$$compliantBottle(X, Z) \leftarrow preferredWine(X, \text{“}WhiteWine\text{”}), wineBottle(Z), DL[\text{“}WhiteWine\text{”}](Z). \tag{12}$$

% Available bottles a person dislikes:

$$doesNotLike(X, Z) \leftarrow person(X), wineBottle(Z), not\ compliantBottle(X, Z). \tag{13}$$

% Generation of multiple answer sets:

$$bottleChosen(X) \vee -bottleChosen(X) \leftarrow compliantBottle(Y, X). \tag{14}$$

% Ensuring that each person gets a bottle:

$$hasBottleChosen(X) \leftarrow bottleChosen(X), compliantBottle(X, Z); \tag{15}$$

$$\leftarrow person(X), not\ hasBottleChosen(X). \tag{16}$$

Fig. 4. dl-program for wine selection

- for $op_i = \uplus$, $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, and
- for $op_i = \cup$, $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$.

We say that I is a *model* of a ground dl-rule r under L, denoted $I \models_L r$, iff $I \models_L H(r)$ whenever $I \models_L l$ for all $l \in B^+(r)$ and $I \not\models_L l$ for all $l \in B^-(r)$. Furthermore, I is a model of a dl-program $KB = (L, P)$, denoted $I \models KB$, iff $I \models_L r$ for all $r \in ground(P)$. We say that KB is *satisfiable* (resp., *unsatisfiable*) iff it has some (resp., no) model.

Note that the herein introduced dl-atoms are *monotonic*: A ground dl-atom a is said to be monotonic whenever given two interpretations $I' \subseteq I''$ it holds that if $I' \models_L a$ then $I'' \models_L a$ as well.

Example 13 (Wine program, OWL, continued). Consider the interpretation

$$I = \{wineBottle(\text{“}TaylorPort\text{”}), preferredWine(\text{“}gibbi\text{”}, \text{“}redWine\text{”}), \\ isA(\text{“}TaylorPort\text{”}, \text{“}redWine\text{”})\},$$

and the rule r, given by:

$$isA(\text{“}TaylorPort\text{”}, \text{“}redWine\text{”}) \leftarrow wineBottle(\text{“}TaylorPort\text{”}), \\ DL[\text{“}RedWine\text{”}](\text{“}TaylorPort\text{”}).$$

Suppose “$RedWine$”(“$TaylorPort$”) is true in the ontology. Then, we have that $I \models_L DL[\text{“}RedWine\text{”}](\text{“}TaylorPort\text{”})$, and hence $I \models_L r$. On the other hand, $I \not\models_L s$, where s is given by

$$compliantBottle(\text{“}gibbi\text{”}, \text{“}TaylorPort\text{”}) \leftarrow preferredWine(\text{“}gibbi\text{”}, \text{“}redWine\text{”}), \\ wineBottle(\text{“}TaylorPort\text{”}), \\ DL[\text{“}RedWine\text{”}](\text{“}TaylorPort\text{”}),$$

since I contains all atoms in the body of s but not $H(s) = compliantBottle(\text{“}gibbi\text{”}, \text{“}TaylorPort\text{”})$. □

Minimal-model semantics of positive dl-programs. We first consider positive dl-programs. Like for ordinary positive programs, every nondisjunctive positive dl-program which is satisfiable has a single minimal model, which naturally characterizes its semantics. Observe that, as pointed out above, dl-atoms considered here are monotonic.

For ordinary normal positive programs P, it is well-known that the intersection of two models of P is also a model of P. A similar result holds for dl-programs.

Theorem 1. *Let $KB = (L, P)$ be a normal positive dl-program. If the interpretations $I_1, I_2 \subseteq HB_P$ are models of KB, then $I_1 \cap I_2$ is also a model of KB.*

As an immediate corollary of this result, every satisfiable positive dl-program KB has a unique least model, denoted M_{KB}, which is contained in every model of KB.

Corollary 1. *Let $KB = (L, P)$ be a normal positive dl-program. If KB is satisfiable, then there is a unique model $I \subseteq HB_P$ of KB such that $I \subseteq J$ for all models $J \subseteq HB_P$ of KB.*

Example 14. Consider Rules (5)–(12) in Fig. 4. Combined with the classical wine ontology, which is consistent, they have a single minimal model. □

On the other hand, if a dl-program contains disjunction, then multiple minimal models of KB may exist.

Example 15. Consider again the program in Fig. 4, and disregard the rules containing default negation “not”. In the wine ontology, each class $RedWine$, $WhiteWine$, and $DryWine$ has several instances (and some of them have common instances, e.g., “$TaylorPort$”). Therefore, for each of $axel$, $gibbi$, and $roman$, multiple possibilities to choose a compliant bottle exist. In combination, they give rise to multiple answer sets of the reduced program. □

Strong answer-set semantics of dl-programs. We now define the *strong answer-set semantics* of general dl-programs. It reduces to the minimal model semantics for positive dl-programs, using a generalized transformation that removes all NAF-literals.

In the sequel, let $KB = (L, P)$ be a dl-program.

Definition 1. *The* strong dl-reduct *of P relative to L and an interpretation $I \subseteq HB_P$, denoted sP_L^I, is the set of all dl-rules obtained from $ground(P)$ by*

(i) *deleting every dl-rule r such that $I \models_L \ell$ for some $\ell \in B^-(r)$, and*
(ii) *deleting from each remaining dl-rule r all literals in $B^-(r)$.*

Note that (L, sP_L^I) is a positive dl-program. Moreover, by Corollary 1, it has a least model if it is satisfiable and normal.

Definition 2. *Let $KB = (L, P)$ be a dl-program. A* strong answer set *of KB is an interpretation $I \subseteq HB_P$ such that I is a minimal model of (L, sP_L^I).*

Example 16 (Wine program, OWL continued). Suppose that the concept $RedWine$ possesses the instances "$TaylorPort$" and "$ChiantiClassico$", $WhiteWine$ the instance "$SelaksIceWine$", and $DryWine$ the instance "$ChateauMargaux$", and assume that $SweetWine$ is empty. Note that these concepts are all subconcepts of $Wine$.

Consider the interpretation I which includes, besides the facts in the program, the following items:

$compliantBottle(\text{"}axel\text{"}, \text{"}SelaksIceWine\text{"})$;
$compliantBottle(\text{"}gibbi\text{"}, \text{"}TaylorPort\text{"})$;
$compliantBottle(\text{"}gibbi\text{"}, \text{"}ChiantiClassico\text{"})$;
$compliantBottle(\text{"}roman\text{"}, \text{"}ChateauMargaux\text{"})$;

$bottleChosen(\text{"}axel\text{"})$; $bottleChosen(\text{"}gibbi\text{"})$; $bottleChosen(\text{"}roman\text{"})$;
$hasBottleChosen(\text{"}axel\text{"})$; $hasBottleChosen(\text{"}gibbi\text{"})$;
$hasBottleChosen(\text{"}roman\text{"})$;

$doesNotLike(\text{"}axel\text{"}, \text{"}TaylorPort\text{"})$;
$doesNotLike(\text{"}axel\text{"}, \text{"}ChiantiClassico\text{"})$;
$doesNotLike(\text{"}axel\text{"}, \text{"}ChateauMargaux\text{"})$;
$doesNotLike(\text{"}gibbi\text{"}, \text{"}SelaksIceWine\text{"})$;
$doesNotLike(\text{"}gibbi\text{"}, \text{"}ChateauMargaux\text{"})$;
$doesNotLike(\text{"}roman\text{"}, \text{"}SelaksIceWine\text{"})$;
$doesNotLike(\text{"}roman\text{"}, \text{"}TaylorPort\text{"})$;
$doesNotLike(\text{"}roman\text{"}, \text{"}ChiantiClassico\text{"})$.

It can be checked that I is a strong answer set of KB. Indeed, I satisfies all positive rules in P, as well as all rules of form

$$doesNotLike(p, w) \leftarrow person(p), wineBottle(w),$$

stemming from Rule (13) in Fig. 4, for each pair p, w such that $compliantBottle(p, w)$ is not contained in I. Furthermore, Rule (16) vanishes in the reduction. Thus, I is a model of (L, sP_L^I). Moreover, I is minimal as no facts can be removed from it without losing modelhood. Therefore, I is an strong answer set of KB. □

The following result shows that the strong answer-set semantics of a dl-program $KB = (L, P)$ conservatively extends the ordinary answer-set semantics of P.

Theorem 2. *Let $KB = (L, P)$ be a dl-program without dl-atoms. Then, $I \subseteq HB_P$ is a strong answer set of KB iff it is an answer set of the ordinary program P.*

As desired, strong answer sets of a dl-program KB are also models, and, moreover, minimal.

Theorem 3. *Let $KB = (L, P)$ be a dl-program and let M be a strong answer set of KB. Then, (a) M is a model of KB, and (b) M is a minimal model of KB.*

Proof. (a) Let I be a strong answer set of KB. To show that I is also a model of KB, we have to show that $I \models_L r$ for all $r \in ground(P)$. Consider any $r \in ground(P)$. Suppose that $I \models_L \ell$ for all $\ell \in B^+(r)$ and $I \not\models_L \ell$ for all $\ell \in B^-(r)$. Then, the dl-rule r' that is obtained from r by removing all the literals in $B^-(r)$ is contained in sP_L^I. Since I is a minimal model of (L, sP_L^I) and thus in particular a model of (L, sP_L^I), it follows that I is a model of r'. Since $I \models_L \ell$ for all $\ell \in B^+(r')$ and $I \not\models_L \ell$ for all $\ell \in B^-(r') = \emptyset$, it follows that $I \models_L \ell'$ for some $\ell' \in H(r)$. This shows that $I \models_L r$. Also, each rule $r \in ground(P)$ having no counterpart in sP_L^I is trivially modeled by I since $I \not\models B(r)$.

Hence, I is a model of KB.

(b) By Part (a), every strong answer set I of KB is a model of KB. We show that I is a minimal model of KB. Towards a contradiction, suppose that there exists a model J of KB such that $J \subset I$. Since J is a model of KB, it follows that J is also a model of (L, sP_L^J). As every dl-atom in DL_P is monotonic relative to KB, it then follows that $sP_L^I \subseteq sP_L^J$. Hence, J is also a model of (L, sP_L^I). But this contradicts that I is a minimal model of (L, sP_L^I). Hence, I is a minimal model of KB. □

Note that every normal positive dl-program KB has at most one strong answer set, which coincides with the single minimal model of KB.

5.3 Further Examples

Closed-world reasoning. As stressed in Section 4, it is acknowledged that many Semantic-Web application scenarios require some form of closed-world reasoning [1, 28].

Using dl-programs, the CWA may be easily expressed on top of an external knowledge base which can be queried through suitable dl-atoms. We show this here for a description-logic knowledge base L.

Intuitively given a concept C, its negated version $\bar{C}$ (under CWA) is defined by adding to a given dl-program the rule

$$\bar{C}(X) \leftarrow not\ DL[C](X)$$

For example, given that $L = \{\, WhiteWine \sqsubseteq Wine,\ \ Wine(\text{“}ChiantiClassico\text{”})\}$, for concepts $WhiteWine$ and $Wine$, the CWA infers $\neg\, WhiteWine(\text{“}ChiantiClassico\text{”})$.

As well known, the CWA can lead to inconsistent conclusions. If, in the above example, L contains further axioms

$$Wine = WhiteWine \sqcup \neg RedWine \ \text{ and}$$
$$\bot = WhiteWine \sqcap \neg RedWine,$$

then the CWA infers

$$\overline{WhiteWine}(\text{“}ChiantiClassico\text{”}) \text{ and } \overline{RedWine}(\text{“}ChiantiClassico\text{”}),$$

which is inconsistent with L.

We can check inconsistency of the CWA with the further rule

$$fail \leftarrow DL[WhiteWine \uplus \overline{WhiteWine}, RedWine \uplus \overline{RedWine}; \bot](b), not\ fail,$$

where $\bot$ is the empty concept (entailment of $\bot(b)$, for any constant b, is tantamount to inconsistency).

Workarounds to these semantic difficulties are well known in the literature: minimal-model reasoning, or the extended closed-world assumption (ECWA), for instance, avoid the problem of CWA inconsistency [9, 23]. These extensions can be easily implemented in the framework of dl-programs, by means of a suitable encoding that computes minimal models of a knowledge base L. Intuitively, building minimal models of L corresponds to concluding as much negative facts as possible while keeping consistency.

Default reasoning. By *maximizing* rather than *minimizing* extensions, default reasoning, as in the approach by Poole [48], on top of a description-logic knowledge base may be supported. The rationale is to associate to individuals default values for concept and roles. Default information is maximized, in the sense that it is propagated as much as possible unless inconsistency arises.

Although acknowledged as being essential for modeling reasoning in the Semantic-Web context (see, e.g., [3]), description-logic knowledge bases do not allow nonmonotonic inheritance. This often causes many ontology design problems, especially in those cases where overriding some default-concept property value is the most natural way of defining a subclass. Defaults are especially tailored at implementing nonmonotonic inheritance. For example, the rules

$$\begin{aligned} shouldbewhite(W) &\leftarrow DL[sparklingWine](W), not\ nonwhite(W), \\ nonwhite(W) &\leftarrow DL[WhiteWine \uplus shouldbewhite; \neg WhiteWine](W) \end{aligned}$$

on top of a part, L, of the wine ontology express that sparkling wines are white by default. Given

$$\begin{aligned} L = \{\ & sparklingWine(\text{“}VeuveCliquot\text{”}), \\ & (sparklingWine \sqcap \neg whiteWine)(\text{“}Lambrusco\text{”})\}, \end{aligned}$$

we then can conclude $white(\text{“}VeuveCliquot\text{”})$ and $nonwhite(\text{“}Lambrusco\text{”})$.

5.4 Additional Features of dl-Programs

An interesting fragment of dl-programs are *stratified dl-programs*, which are, intuitively, composed of hierarchic layers of positive dl-programs linked via default negation. This generalization of the classic notion of stratification embodies a fragment of the language having single answer sets. Semantics for programs (or sub-programs) belonging to this fragment can be evaluated at a less expensive computational cost [15].

Furthermore, it is possible to evaluate dl-programs either under *weak answer-set semantics* [18] and a *well-founded semantics* [19]. The former does not make any assumption on the nature of a dl-atom (whereas *monotonic* dl-atoms are treated explicitly in the semantics discussed here), while the latter is a generalization of the traditional well-founded semantics [56] for dl-programs.

5.5 Prototype Implementation

A fully operational prototype, named *NLP-DL*, ready for experiments, is available via a Web interface at

`http://www.kr.tuwien.ac.at/staff/roman/semweblp/`

The system accepts nondisjunctive dl-programs as input,[9] given by an ontology formulated in OWL DL (as processed by RACER [26]) and a set of dl-rules in the language above, where $\leftarrow$, $\uplus$, and $\cup$, are written as ":-", "+=", and "–=", respectively. The following reasoning tasks are featured:

(i) *Computing models* (*answer sets or the well-founded model*) *of a given dl-program:* For computing the answer sets, a preliminary computation of the well-founded model may be issued, which semantically approximates the answer sets—this is exploited for optimization.
(ii) *Evaluating a given query on a given dl-program:* Under the answer-set semantics, both *brave reasoning* and *cautious reasoning* are available.

The system architecture integrates the external DLV [36] and RACER engines, the latter being embedded into a caching module, a well-founded semantics module, an answer-set semantics module, a pre-processing module, and a post-processing module.

Each internal module has been implemented using the PHP scripting language; the overhead is insignificant, provided that most of the computing power is devoted to the execution of the two external reasoners. In particular, efficient usage of RACER is critical for the system performance. Respective techniques, mainly based on caching query results and exploiting monotonicity of description-logic reasoning, are described in [15].

6 Extensions

Example 17 (Motivating Example, Part III). Now that a machinery, automatically generating a selection of wine bottles for the social dinner, is ready, the organizers wonder whether it is possible to accomplish this task in a better way. After all, the Semantic Web envisions a world where machine-to-machine protocols express their full potential, and people are freed from most annoying jobs. In this context, multiple domain descriptions (i.e., multiple ontologies), possibly with differing semantics, may interact closely and have to be ready for information exchange.

[9] An implementation of disjunctive dl-programs is available through dlvhex, an implementation of HEX-programs (see next section for details about HEX-programs and dlvhex).

For instance, most of the attendees may have his or her own FOAF [21] description on-line. These description might potentially publish all the public data about an attendee, including his or her preferred wine. However, now the organizers notice that they need some formalism powerful enough to interface several formalisms and multiple ontologies at once. □

6.1 HEX-programs

HEX-programs generalize dl-programs with regard to the following features:

- The notion of a dl-atom is generalized to that of an *external atom*. The latter kind of atom may bind knowledge coming from different external formalisms, with possibly differing semantics. Also, an external atom can delegate special tasks to traditional programs (such as string processing), for which logic programming is not tailored at. For instance, it is possible to merge RDF ontologies with OWL ontologies, as in the following small program:

$$\begin{aligned} triple(X,Y,Z) &\leftarrow url(U), \&rdf[U](X,Y,Z); \\ &\leftarrow \&DLinconsistent[triple]. \end{aligned}$$

  Also, possible external sources of knowledge can be merged with arbitrary strategies, and can bring in new symbols not appearing elsewhere in a given program ("value invention").
- It is made possible to quantify over sets of concepts just as it is done with individuals, and to freely exchange the former objects with the latter ones. These meta-reasoning features are enabled by means of *higher-order atoms*, such as in the rule

$$\text{“wine:Wine”}(X) \leftarrow triple(X, \text{“rdf:type”}, \text{“wine:Wine”}).$$

- Logic programs are made compatible with naming conventions employed in the Semantic-Web world. Thus, a directive such as

$$\#namespace(wine, \text{“}\texttt{http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine\#}\text{”})$$

  allows to interpret the constant symbol "wine:Wine" as a shortcut for the symbol

  "`http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Wine`".

In this section, we briefly discuss HEX-programs; for further details, see [14].

6.2 Syntax and Semantics

HEX-programs are built on mutually disjoint sets $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ of *constant names*, *variable names*, and *external predicate names*, respectively. Unless stated otherwise, elements from $\mathcal{X}$ (resp., $\mathcal{C}$) are denoted with first letter in upper case (resp., lower case);

elements from $\mathcal{G}$ are prefixed with "&".[10] Constant names serve both as individual and predicate names. Importantly, $\mathcal{C}$ may be infinite.

Elements from $\mathcal{C} \cup \mathcal{X}$ are called *terms*. A *higher-order atom* (or *atom*) is a tuple $(Y_0, Y_1, \ldots, Y_n)$, where $Y_0, \ldots, Y_n$ are terms; $n \geq 0$ is its *arity*. Intuitively, Y_0 is the predicate name; we thus also use the familiar notation $Y_0(Y_1, \ldots, Y_n)$. The atom is *ordinary*, if Y_0 is a constant. For example, $(x, rdf{:}type, c)$ and $node(X)$ are ordinary atoms, while $D(a, b)$ is a higher-order atom. An *external atom* is of the form

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m),$$

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two lists of terms (called *input list* and *output list*, respectively), and $\&g \in \mathcal{G}$ is an external predicate name. We assume that $\&g$ has fixed lengths $in(\&g) = n$ and $out(\&g) = m$, respectively. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates.

Example 18. The external atom $\&reach[edge, a](X)$ may compute the nodes reachable in the graph $edge$ from the node a. Here, $in(\&reach) = 2$ and $out(\&reach) = 1$. □

A HEX-*program*, P, is a finite set of rules of form (1), where literals in the heads of rules are (higher-order) atoms, and literals in the bodies of rules contain either (higher-order) atoms or external atoms.

The semantics of HEX-programs generalizes the answer-set semantics [22], and is defined using the *FLP-reduct* [20], which is more elegant than the traditional reduct and ensures minimality of answer sets.

The *Herbrand base* of a HEX-program P, denoted HB_P, is the set of all possible ground versions of atoms and external atoms occurring in P obtained by replacing variables with constants from $\mathcal{C}$. The grounding of a rule r, $ground(r)$, is defined accordingly, and the grounding of program P is $ground(P) = \bigcup_{r \in P} ground(r)$.

Example 19. For $\mathcal{C} = \{edge, arc, a, b\}$, ground instances of $E(X, b)$ are, for instance, $edge(a, b)$, $arc(a, b)$, and $arc(arc, b)$; ground instances of $\&reach[edge, N](X)$ are $\&reach[edge, edge](a)$, $\&reach[edge, arc](b)$, and $\&reach[edge, edge](edge)$, etc. □

An *interpretation relative to* P is any subset $I \subseteq HB_P$ containing only atoms. We say that I is a *model* of atom $a \in HB_P$, denoted $I \models a$, if $a \in I$. With every external predicate name $\&g \in \mathcal{G}$ we associate an $(n{+}m{+}1)$-ary Boolean function $f_{\&g}$ (called *oracle function*) assigning each tuple $(I, y_1 \ldots, y_n, x_1, \ldots, x_m)$ either 0 or 1, where $n = in(\&g)$, $m = out(\&g)$, $I \subseteq HB_P$, and $x_i, y_j \in \mathcal{C}$. We say that $I \subseteq HB_P$ is a *model* of a ground external atom $a = \&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$, denoted $I \models a$, iff $f_{\&g}(I, y_1 \ldots, y_n, x_1, \ldots, x_m) = 1$.

Example 20. Associate with the external predicate name $\&reach$ a function $f_{\&reach}$ such that $f_{\&reach}(I, E, A, B) = 1$ iff B is reachable in the graph E from A. Let $I = \{e(b, c), e(c, d)\}$. Then, I is a model of the external atom $\&reach[e, b](d)$ since $f_{\&reach}(I, e, b, d) = 1$. □

[10] In [14], "#" is used instead of "&"; the change is motivated to be in accord with the syntax of the prototype system.

Let r be a ground rule. We define (i) $I \models H(r)$ iff there is some $a \in H(r)$ such that $I \models a$, (ii) $I \models B(r)$ iff $I \models a$ for all $a \in B^+(r)$ and $I \not\models a$ for all $a \in B^-(r)$, and (iii) $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$. We say that I is a *model* of a HEX-program P, denoted $I \models P$, iff $I \models r$ for all $r \in ground(P)$.

The *FLP-reduct* [20] of P with respect to $I \subseteq HB_P$, denoted fP^I, is the set of all $r \in ground(P)$ such that $I \models B(r)$. $I \subseteq HB_P$ is an *answer set of* P iff I is a minimal model of fP^I.

Differences between the FLP-reduct and the strong dl-reduct. The two above semantics are not equivalent in the presence of nonmonotonic external atoms, where the notion of monotonicity for an external atom generalizes that for dl-atoms. Let us assume to have an external predicate $\&neg$, defined in such a way that the ground atom $\&neg[p](a)$ satisfies $I \not\models \&neg[p](a)$ whenever an interpretation I is such that $I \models p(a)$ (i.e., $\&neg$ reproduces the behavior of the usual negation as failure). The program P, consisting of the single rule

$$p(a) \leftarrow not\ \&neg[p](a),$$

has $S_1 = \{p(a)\}$ as a strong answer set. However, also $S_2 = \emptyset$ is a strong answer set of P, thus S_1 is not minimal. It is often desirable that answer sets are incomparable as in the above case: intuitively, self-supportedness of an atom such as in the rule $p(a) \leftarrow p(a)$ should not give evidence of the truth of $p(a)$.

The FLP-reduct overcomes these drawbacks. Indeed, it can be proven that this reduct produces only incomparable answer sets: under FLP semantics, S_1 is not an answer set.

6.3 Further Examples

With HEX-programs, it is possible to extract information from different sources in the same program.

Assume we want to invite all friends of Axel Polleres for dinner, and that their wine preferences are given by means of their FOAF descriptions. To this end, we introduce the $\&rdf$ atom for dealing with RDF sources, and the $\&dlC$ atom that mimics partially the semantics of a dl-atom. An atom $\&rdf[u](s, p, o)$ is true if $\langle s\ p\ o \rangle$ is an RDF triple asserted at URI u. Also, $\&dlC[u, c](x)$ is true if x is an individual which can be proved to belong to class c in the knowledge base located at URI u (under OWL semantics).

First, namespace directives allow us to deal with individuals and concepts (constant symbols) coming from different Web sources:

$\#namespace(wine,$ "`http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#`");

$\#namespace(foaf,$ "`http://xmlns.com/foaf/0.1/`");

$\#namespace(rdf,$ "`http://www.w3.org/1999/02/22-rdf-syntax-ns#`");

$\#namespace(foafplus,$ "`http://www.example.org/foafplus#`");

$\#namespace(rdfs,$ "`http://www.w3.org/2000/01/rdf-schema#`").

```
<foaf:PersonalProfileDocument rdf:about="">
  <foaf:maker rdf:resource="#me"/>
  <foaf:primaryTopic rdf:resource="#me"/>
  ...
</foaf:PersonalProfileDocument>

  <foaf:Person rdf:ID="me">
    <foaf:name>Axel Polleres</foaf:name>
    ...
    <foaf:knows>
      <foaf:Person>

        <foaf:name>Giovambattista Ianni</foaf:name>
        <foaf:mbox>ianni@mat.unical.it</foaf:mbox>
        <rdfs:seeAlso rdf:resource=
         "http://www.gibbi.com/test_foaf.gibbi.rdf"/>
      </foaf:Person>
    </foaf:knows>
        ...
    <foafplus:winePreference rdf:resource="&vin;SweetWine"/>
  </foaf:Person>
```

Fig. 5. An example FOAF description, extended with the foafplus:winePreference property

Suppose now that a FOAF description is given, like in Fig. 5. This FOAF description is enriched with the property `foafplus:winePreference` which expresses a wine preference for a given person. This small description can be interfaced with a HEX-program in the following way:

$$Y(X, Z, triple) \leftarrow \&rdf[U](X, Y, Z), foafurl(U);$$
$$T(X, triple) \leftarrow \text{“rdf:type”}(X, T, triple).$$

The above rules materialize the RDF triples contained in Axel's FOAF description.

Then, the predicate $preferredWine$ is now computed by extracting data from external descriptions of Axel's friends (note that further external ontologies are consulted whose locations depend on the first consulted ontology):

$$\begin{aligned} mainEntity(M, triple) &\leftarrow \text{“foaf:primaryTopic”}(X, M, triple), \\ &\quad \text{“foaf:PersonalProfileDocument”}(X, triple); \\ community(A, Y) &\leftarrow \text{“foaf:knows”}(X, A, triple), \\ &\quad \text{“rdfs:seeAlso”}(A, Y, triple); \\ preferredWine(M, Y) &\leftarrow \text{“foafplus:winePreference”}(M, Y, triple), \\ &\quad mainEntity(M, triple); \\ preferredWine(X, Y) &\leftarrow community(X, U), \\ &\quad \&rdf[U](\text{“foafplus:winePreference”}, Y). \end{aligned}$$

The next rule facilitates the quantification over concept names given to the predicate `&dlC`:

$$compliantBottle(X, Z) \leftarrow wineurl(U), preferredWine(X, Y), \\ \&dlC[U, Y](Z).$$

Note that this rule allows to generalize, for instance, Rules (9)–(12) of the program given in Fig. 4. The rest of the program is very similar to the latter one:

$$bottleChosen(X) \vee -bottleChosen(X) \leftarrow compliantBottle(Y, X);$$
$$hasBottleChosen(X) \leftarrow bottleChosen(Z), compliantBottle(X, Z);$$
$$\leftarrow preferredWine(X, Y), not\ hasBottleChosen(X);$$
$$:\sim\ bottleChosen(X)\ [1].$$

6.4 Prototype Implementation

An experimental prototype for evaluating HEX-programs, called dlvhex, is available and executable on the Web at

`http://www.kr.tuwien.ac.at/research/dlvhex/`

Apart from implementing the semantics of HEX-programs, dlvhex supports a number of built-in functions as well as integrity and weak constraints. Its further development is work in progress.

The principle behind dlvhex is to represent a framework that integrates a native answer-set solver—here, DLV [36]—and the external reasoners underlying the external atoms. Optionally, dlvhex can integrate DLT [10] as a pre-parser to allow for templates and frame syntax within HEX-programs. Due to the bidirectional nature of external atoms, they cannot be evaluated prior to calling the answer-set solver. Instead, dlvhex builds the dependency graph of the HEX-program, identifying minimal sets of nodes that involve external atoms, which have to be solved by specifically tailored algorithms. This strategy, which is described in more detail in [16] and [17], relies basically on a modified version of the well-known splitting-set theorem for ordinary logic programs [40].

The evaluation functions of the external atoms are defined completely independent from dlvhex by so called *plug-ins*, which can contain the implementations of several atoms. The currently available external atoms are the *RDF Plug-in*, the *Description-Logics Plug-in* and the *String Plug-in*, described below.

The RDF Plug-in The RDF plug-in currently provides a single external atom, the $\&rdf$ atom, which enables the user to import RDF triples from any RDF knowledge base. It takes a single constant as input, which denotes the RDF source (a file path or a Web address). The $\&rdf$ atom interfaces with the RAPTOR RDF library.

The Description-Logics Plug-in In order to model dl-programs [18] in terms of HEX-programs, the Description-Logics Plug-in has been developed. This plug-in includes three external atoms (these atoms, in accord to the semantics of dl-programs, also allow for extending a description-logic knowledge base, before submitting a query, by means of the atoms' input parameters):

- the $\&dlC$ atom, which queries a concept (specified by an input parameter of the atom) and retrieves its individuals;
- the $\&dlR$ atom, which queries an object property and retrieves its individual pairs;
- the $\&dlDR$ atom, which queries a datatype property and retrieves its pairs; and
- the $\&dlConsistent$ atom, which tests the (possibly extended) description-logic knowledge base for consistency.

The Description-Logics Plug-in can access OWL ontologies, i.e., description-logic knowledge bases in the language $\mathcal{SHOIN}(\mathbf{D})$, utilizing the RACER [26] reasoning engine.

The String Plug-in The task of the String Plug-in is to realize simple string manipulations.

Currently, dlvhex, together with the presented plug-ins, are available as source packages. Moreover, a toolkit for developing custom plug-in is supplied as well, embedded in the GNU auto-tools environment, which takes care for the low-level, system-specific build process and which allows the plug-in author to concentrate his or her efforts on the implementation of the plug-in's actual core functionality.

7 Discussion and Conclusion

We have considered reasoning with rules and ontologies, taking an answer-set programming perspective. A number of approaches for combining rules and ontologies have been presented so far, and the quest for the Holy Grail of an ideally suited formalism (which might not exist) is still ongoing. As we have briefly discussed, a number of issues come up when combining rules as in logic programming and ontologies formalized in classical logic. Bridging the quite different worlds of logic programs and ontologies has been attempted in different approaches, which may be grouped in "tightly" coupled and "loosely" coupled approaches.

The approach which is closest in spirit to dl-programs is Rosati's $\mathcal{DL}{+}log$ formalism [52, 53], which extends his previous work [50, 51]. In this approach, predicates are split into *ontology predicates* and into *logic-program* (*datalog*) *predicates*. A notion of model of a combined rule and ontology knowledge base is defined using a two-step reduct in which, in the first step, the ontology predicates are eliminated under the open-world assumption (OWA) and, in the second step, the negated logic-programming predicates under the closed-world assumption (CWA). As shown by Rosati, the emerging formalism (which focuses on first-order models under the standard-names assumption), is decidable provided that conjunctive-query answering over the underlying ontology is decidable. The main differences between $\mathcal{DL}{+}log$ and dl-programs are as follows:

- $\mathcal{DL}{+}log$ is a tight coupling, while dl-programs provide a loose coupling of rules and ontologies.
- While extensions of dl-programs to integrate ontologies even in different formats are straightforward, there is no corresponding counterpart in $\mathcal{DL}{+}log$.

– The approach of dl-atoms is more flexible for mixing different reasoning modalities, such as consistency checking and logical consequence. In the realm of HEX-programs, almost arbitrary combinations can be conceived.
– The coupling as realized in dl-programs aims at facilitating interoperability of existing reasoning systems and software (such as DLV and RACER). On the other hand, the loose coupling requires a bridging between the two worlds of ontologies and rules, which has to be provided by the user. In particular, this applies to the individuals at the instance level.

The development and theoretical study of HEX-programs is ongoing. Algorithms and techniques for efficient implementation are in an advanced stage of progression. In a sense, rules are per se a form or knowledge that needs to be exchanged and evaluated under different semantics. To this end, we are developing an exchange format aimed at fitting answer-set programming in the RuleML standard. In conclusion, although quite some efforts have been spent on combining rules and ontologies, there is still a lot of work to be done.

References

1. A. Analyti, G. Antoniou, C. V. Damásio, and G. Wagner. Stable Model Theory for Extended RDF Ontologies. In *Proc. Fourth International Semantic Web Conference* (*ISWC 2005*), pp. 21–36, 2005.
2. J. Angele, H. Boley, J. de Bruijn, D. Fensel, P. Hitzler, M. Kifer, R. Krummenacher, H. Lausen, A. Polleres, and R. Studer. Web Rule Language (WRL), Sept. 2005. W3C Member Submission, `http://www.w3.org/Submission/WRL/`.
3. G. Antoniou. Nonmonotonic Rule Systems on Top of Ontology Layers. In *Proc. First International Semantic Web Conference* (*2002*), volume 2342 of *Lecture Notes in Computer Science* (*LNCS*), pp. 394–398, 2002.
4. G. Antoniou, C. V. Damásio, B. Grosof, I. Horrocks, M. Kifer, J. Maluszynski, and P. F. Patel-Schneider. Combining Rules and Ontologies: A survey. Technical Report IST506779/Linköping/I3-D3/D/PU/a1, Linköping University, February 2005. IST-2004-506779 REWERSE Deliverable I3-D3. `http://rewerse.net/publications/`.
5. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
6. C. Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, Cambridge, UK, 2003.
7. T. Berners-Lee. Web for Real People, April 2005. Keynote Speech at the 14th World Wide Web Conference (WWW2005). Slides available at `http://www.w3.org/2005/Talks/0511-keynote-tbl/`.
8. F. Buccafurri, N. Leone, and P. Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.
9. M. Cadoli and M. Lenzerini. The Complexity of Propositional Closed World Reasoning and Circumscription. *Journal of Computer and System Sciences*, 43:165–211, April 1994.
10. F. Calimeri, G. Ianni, G. Ielpa, A. Pietramala, and M. C. Santoro. A System with Template Answer Set Programs. In *Proc. Ninth European Conference on Artificial Intelligence* (*JELIA 2004*), volume 3229 of *Lecture Notes in AI* (*LNAI*), pp. 693–697. Springer Verlag, 2004.

11. M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference, Feb. 2004. W3C Recommendation.
12. F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. $\mathcal{AL}$-log: Integrating Datalog and Description Logics. *Journal of Intelligent Information Systems (JIIS)*, 10(3):227–252, 1998.
13. The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05) contest. `http://www.comp.hkbu.edu.hk/~eee05/contest/`.
14. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In *Proc. 19th International Joint Conference on Artificial Intelligence* (*IJCAI 2005*). Morgan Kaufmann, 2005.
15. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Nonmonotonic Description Logic Programs: Implementation and Experiments. In F. Baader and A. Voronkov, editors, *Proc. Eleventh International Conference on Logic for Programming, Artificial Intelligence, and Reasoning* (*LPAR 2004*), number 3452 in LNCS, pp. 511–527. Springer, 2005.
16. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Effective Integration of Declarative Rules with External Evaluations for Semantic Web Reasoning. In Y. Sure and J. Domingue, editors, *Proc. Third European Semantic Web Conference* (*ESWC 2006*), number 4011 in LNCS, pp. 273–287. Springer, 2006.
17. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Towards Efficient Evaluation of HEX Programs. In J. Dix and A. Hunter, editors, *Proc. Eleventh International Workshop on Nonmonotonic Reasoning* (*NMR 2006*)*, Answer Set Programming Track*, pp. 40–46, 2006.
18. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. In *Proc. Ninth International Conference on Principles of Knowledge Representation and Reasoning* (*KR2004*), pp. 141–151, 2004.
19. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Well-Founded Semantics for Description Logic Programs in the Semantic Web. In *Proc. ISWC 2004 Workshop on Rules and Rule Markup Languages for the Semantic Web* (*RuleML 2004*), volume 3323 of *Lecture Notes in Computer Science* (*LNCS*), pp. 81–97. Springer Verlag, 2004.
20. W. Faber, N. Leone, and G. Pfeifer. Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. In *Proc. Ninth European Conference on Artificial Intelligence* (*JELIA 2004*), number 3229 in Lecture Notes in AI (LNAI), pp. 200–212. Springer Verlag, 2004.
21. The Friend of a Friend (FOAF) Project. `http://www.foaf-project.org/`.
22. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
23. M. Gelfond, H. Przymusinska, and T. C. Przymusinski. The Extended Closed World Assumption and its Relationship to Parallel Circumscription. In *Proc. Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (*PODS '86*), pp. 133–139, 1986.
24. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logics. In *Proc. Twelfth International World Wide Web Conference* (*WWW 2003*), pp. 48–57, 2003.
25. T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5:199–220, 1993.
26. V. Haarslev and R. Möller. RACER System Description. In *Proc. First International Joint Conference on Automated Reasoning* (*IJCAR 2001*), volume 2083 of *Lecture Notes in Computer Science* (*LNCS*), pp. 701–705. Springer Verlag, 2001.
27. P. Hayes. RDF semantics. `http://www.w3.org/TR/rdf-mt/`.
28. J. Heflin and H. Munoz-Avila. LCW-Based Agent Planning for the Semantic Web. In *Proc. AAAI Workshop on Ontologies and the Semantic Web*, pp. 63–70, 1998.

29. S. Heymans, D. V. Nieuwenborgh, and D. Vermeir. Nonmonotonic Ontological and Rule-Based Reasoning with Extended Conceptual Logic Programs. In *Proc. Second European Semantic Web Conference* (*ESWC 2005*), volume 3532 of *Lecture Notes in Computer Science* (*LNCS*), pp. 392–407. Springer Verlag, 2005.
30. S. Heymans, D. V. Nieuwenborgh, and D. Vermeir. Preferential Reasoning on a Web of Trust. In *Proc. Fourth International Semantic Web Conference* (*ISWC 2005*), volume 3729 of *Lecture Notes in Computer Science* (*LNCS*), pp. 368–382. Springer Verlag, 2005.
31. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, May 2004. W3C Member Submission. `http://www.w3.org/Submission/SWRL/`.
32. I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–264, 2000.
33. ICONS homepage, since 2001. `http://www.icons.rodan.pl/`.
34. T. Janhunen, I. Niemelä, D. Seipel, P. Simons, and J.-H. You. Unfolding Partiality and Disjunctions in Stable Model Semantics. *ACM Transactions on Computational Logic*, 7(1):1–37, 2006.
35. N. Leone, G. Gottlob, R. Rosati, T. Eiter, W. Faber, M. Fink, G. Greco, G. Ianni, E. Kałka, D. Lembo, M. Lenzerini, V. Lio, B. Nowicki, M. Ruzzi, W. Staniszkis, and G. Terracina. The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In *Proc. 24th ACM SIGMOD International Conference on Management of Data* (*SIGMOD 2005*), pp. 915–917. ACM Press, 2005.
36. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 2005. To appear. Available at `http://www.arxiv.org/ps/cs.AI/0211004`.
37. A. Y. Levy and M.-C. Rousset. Combining Horn Rules and Description Logics in CARIN. *Artificial Intelligence*, 104(1-2):165–209, 1998.
38. Y. Lierler. Disjunctive Answer Set Programming via Satisfiability. In *Proc. Eighth International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2005*), volume 3662 of *Lecture Notes in Computer Science* (*LNCS*), pp. 447–451. Springer Verlag, 2005.
39. V. Lifschitz. Answer Set Planning. In *Proc. 16th International Conference on Logic Programming* (*ICLP '99*), pp. 23–37. MIT Press, 1999.
40. V. Lifschitz and H. Turner. Splitting a Logic Program. In *Proc. Eleventh International Conference on Logic Programming* (*ICLP '94*), pp. 23–38. MIT Press, 1994.
41. T. Lukasiewicz. Stratified Probabilistic Description Logic Programs. In *Proc. ISWC 2005 Workshop on Uncertainty Reasoning for the Semantic Web*, pp. 87–97, 2005.
42. W. Marek and M. Truszczyński. Stable Logic Programming - An Alternative Logic Programming Paradigm. In K. Apt, W. Marek, and M. Truszczyński, editors, *The Logic Programming Paradigm*, pp. 375–398. Springer Verlag, 1999.
43. B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, 2005.
44. I. Niemelä. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.
45. I. Niemelä, P. Simons, and T. Soininen. Stable Model Semantics of Weight Constraint Rules. In *Proc. Fifth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '99)*, volume 1730 of *Lecture Notes in AI* (*LNAI*), pp. 107–116. Springer Verlag, 1999.
46. J. Z. Pan, E. Franconi, S. Tessaris, G. Stamou, V. Tzouvaras, L. Serafini, I. R. Horrocks, and B. Glimm. Specification of Coordination of Rule and Ontology Languages. Project Deliverable D2.5.1, KnowledgeWeb NoE, June 2004.

47. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax, Feb. 2004. W3C Recommendation.
48. D. Poole. A Logical Framework for Default Reasoning. *Artificial Intelligence*, 36:27–47, 1988.
49. A. Rainer. Web Service Composition under Answer Set Programming. In *Proc. KI 2005 Workshop "Planen, Scheduling und Konfigurieren, Entwerfen"* (*PuK 2005*), 2005.
50. R. Rosati. Towards Expressive KR Systems Integrating Datalog and Description Logics: Preliminary Report. In *Proc. 1999 International Workshop on Description Logics* (*DL '99*), volume 22 of *CEUR Workshop Proceedings*, pp. 160–164. CEUR-WS.org, 1999.
51. R. Rosati. On the Decidability and Complexity of Integrating Ontologies and Rules. *Journal of Web Semantics*, 3(1):61–73, 2005.
52. R. Rosati. $\mathcal{DL}$+*log*: Tight Integration of Description Logics and Disjunctive Datalog. In *Proc. Tenth International Conference on Principles of Knowledge Representation and Reasoning* (*KR 2006*), pp. 68–78. AAAI Press, 2006.
53. R. Rosati. Reasoning with Rules and Ontologies. In P. Barahona, F. Bry, E. Franconi, U. Sattler, and N. Henze, editors, *Reasoning Web, Second International Summer School 2006, Lissabon, Portugal, September 25-29, 2006, Tutorial Lectures*, Lecture Notes in Computer Science (LNCS). Springer Verlag, 2006. This volume.
54. M. Sintek and S. Decker. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *Proc. First International Semantic Web Conference* (*ISWC 2002*), volume 2342 of *Lecture Notes in Computer Science* (*LNCS*), pp. 364–378, 2002.
55. D. Tsarkov and I. Horrocks. Fact++ Description Logic Reasoner: System Description. In *Proc. Third International Joint Conference on Automated Reasoning* (*IJCAR 2006*), 2006.
56. A. Van Gelder, K. A. Ross, and J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.
57. W3C. The Resource Description Framework. `http://www.w3.org/RDF/`.
58. K. Wang, G. Antoniou, R. W. Topor, and A. Sattar. Merging and Aligning Ontologies in dl-Programs. In *Proc. First International Conference on Rules and Rule Markup Languages for the Semantic Web* (*RuleML 2005*), pp. 160–171, 2005.
59. K. Wang, D. Billington, J. Blee, and G. Antoniou. Combining Description Logic and Defeasible Logic for the Semantic Web. In *Proc. ISWC 2004 Workshop on Rules and Rule Markup Languages for the Semantic Web* (*RuleML 2004*), volume 3323 of *Lecture Notes in Computer Science* (*LNCS*), pp. 170–181. Springer Verlag, 2004.
60. ASPLIB: The Answer Set Programming Satisfiability Library. `http://dit.unitn.it/~wasp/Solvers/index.html`.
61. WASP homepage, since 2002. `http://wasp.unime.it/`.
62. The Wine Ontology. `http://www.w3.org/TR/owl-guide/wine.rdf`.
63. S. Woltran. Answer Set Programming: Model Applications and Proofs-of-Concept. Technical Report WP5, Working Group on Answer Set Programming (WASP, IST-FET-2001-37004), July 2005. Available at `http://www.kr.tuwien.ac.at/projects/WASP/report.html`.