1

Hybrid Reasoning with Rules and Ontologies

Włodzimierz Drabent[1,2], Thomas Eiter[3], Giovambattista Ianni[3,4], Thomas Krennwallner[3], Thomas Lukasiewicz[5,3], and Jan Małuszyński[2]

[1] Institute of Computer Science, Polish Academy of Sciences,
ul. Ordona 21, PL – 01-237 Warszawa, Poland
[2] Department of Computer and Information Science, Linköping University,
SE – 581 83 Linköping, Sweden
[3] Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
[4] Dipartimento di Matematica, Università della Calabria
P.te P. Bucci, Cubo 30B, I-87036 Rende, Italy
[5] Computing Laboratory, University of Oxford
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

Summary. The purpose of this chapter is to report on work that has been done in the REWERSE project concerning hybrid reasoning with rules and ontologies. Two major streams of work have been pursued within REWERSE. They start from the predominant semantics of non-monotonic rules in logic programming. The one stream was an extension of non-monotonic logic programs under answer set semantics, with query interfaces to external knowledge sources. The other stream, in the spirit of the $\mathcal{AL}$-log approach of enhanced deductive databases, was an extension of Datalog (with the well-founded semantics, which is predominant in the database area). The former stream led to so-called non-monotonic dl-programs and HEX-programs, and the latter stream to hybrid well-founded semantics. Further variants and derivations of the formalisms (like a well-founded semantics for dl-programs, respecting probabilistic knowledge, priorities, etc.) have been conceived.

1.1 Introduction

The purpose of this chapter is to report on the work that has been done in REWERSE on hybrid reasoning with rules and ontologies. The importance of rules and ontologies for Web applications is reflected by the World Wide Web Consortium's[1] (W3C) proposal of the layered architecture of the Semantic Web, including the ontology layer and the rule layer. The ontology layer of the Semantic Web was quite developed already at the REWERSE start in 2004. In the same year, W3C adopted the Web Ontology Language (OWL) recommendation [32].

[1] `http://www.w3.org/`

On the other hand, the rule layer was a topic addressed by many researchers but was not yet official subject of W3C activities.

Integration of the rule layer with the ontology layer is necessary for rule-based applications using ontologies, like data integration applications. It can be achieved by combining existing ontology languages with existing rule languages, or by defining new languages, expressive enough to define ontologies, rules and their interaction. An important issue in combination of ontology languages and rule languages based on logics is the semantics of the combined language, as a foundation for development of sound reasoners. The REWERSE work reported in this chapter focused on *hybrid* reasoning, where the reasoner of the combined language reuses the existing reasoners of the component ontology language and rule language.

Motivated by the need for hybrid reasoning with rules and ontologies, two major streams of work have been pursued within REWERSE. They start from the predominant semantics of non-monotonic rules in logic programming. The one stream was an extension of non-monotonic logic programs under answer set semantics, with query interfaces to external knowledge sources. The other stream, in the spirit of the $\mathcal{AL}$-log [33] approach of enhanced deductive databases, was an extension of Datalog (with the well-founded semantics, which is predominant in the database area). The former stream lead to so-called non-monotonic dl-programs and HEX-programs, and the latter stream to hybrid well-founded semantics. Further variants and derivations of the formalisms (like a well-founded semantics for dl-programs, respecting probabilistic knowledge, priorities, etc.) have been conceived.

To put the REWERSE work in a broader perspective, the chapter begins with a concise introduction to the *Resource Description Framework* (RDF) layer, which sets the standard for the data model for the Semantic Web, to the RDF Schema, seen as a simple ontology language, and to OWL. We then discuss rule languages considered in integration proposals and present a classification of the major approaches to integration which uses the terminology of [4, 81]. The remaining part of the chapter surveys the REWERSE work on hybrid integration of rules and ontologies.

1.2 Overview of Approaches

This section gives a brief survey of the approaches to combine or integrate reasoning with rules and ontologies on the Web. It starts with a brief introduction to the underlying formalisms of the Semantic Web, followed by discussion on the rule languages considered in integration proposals. Finally, a classification of the integration proposals is presented. For a more comprehensive survey, the interested reader is referred to [40].

1.2.1 RDF and RDF Schema

The Resource Description Framework (RDF) defines the data model for the Semantic Web as labeled, directed graphs. An RDF dataset (that is, an *RDF graph*) can be viewed as a set of the edges of such a graph, commonly represented by *triples* (or *statements*) of the form:

Subject Predicate Object

where

- the edge links *Subject*, which is a *resource* identified by a URI or a *blank node*, to *Object*, which is either another resource, a blank node, a *datatype literal*, or an *XML literal*;
- *Predicate*, in RDF terminology referred to as *property*, is the edge label.

The next example, originating from [40], illustrates the main concepts of RDF.

Example 1. Take a scenario in which three persons named Alice, Bob, and Charles, have certain relationships among each other: Alice knows both Bob and Charles, Bob just knows Charles, and Charles knows nobody.

For encoding the information that "a person called Bob knows a person called Charles" we need a vocabulary including concepts like "person" and "name". We can adopt the so-called FOAF (friend-of-a-friend) RDF vocabulary [84]. Then the statement can be given by the following RDF triples:

`_:b rdf:type foaf:Person`, `_:b foaf:name "Bob"`, `_:b foaf:knows _:c`, `_:c rdf:type foaf:Person`, and `_:c foaf:name "Charles"`,

where the qualified names like `foaf:Person` are shortcuts for full URIs like `http://xmlns.com/foaf/0.1/Person`, making usage of *namespace prefixes* from XML, for ease of legibility. For instance, the triple

```
_:b foaf:name "Bob"
```

expresses that "someone has the name Bob." `_:b` is a blank node and can be seen as an anonymous identifier. In fact, the name for a blank node is meaningful only in the context of a given RDF graph; conceptually, blank node names can be uniformly substituted inside an RDF graph without changing the meaning of the encoded knowledge.

RDF information can be represented in different formats. One of the most common is the RDF/XML syntax.[2]. The much simpler Turtle[3] representation is adopted in SPARQL, the W3C standard language for querying RDF data. The information of the example can be encoded in Turtle as follows:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
_:a rdf:type foaf:Person .
_:a foaf:name "Alice" .
```

[2] `http://www.w3.org/TR/rdf-syntax-grammar/`
[3] `http://www.w3.org/TeamSubmission/turtle/`

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
<http://www.mat.unical.it/~ianni/foaf.rdf>
        a foaf:PersonalProfileDocument.
<http://www.mat.unical.it/~ianni/foaf.rdf> foaf:maker _:me .
<http://www.mat.unical.it/~ianni/foaf.rdf> foaf:primaryTopic _:me .
_:me a foaf:Person .
_:me foaf:name "Giovambattista Ianni" .
_:me foaf:homepage <http://www.gibbi.com> .
_:me foaf:phone <tel:+39-0984-496430> .
_:me foaf:knows [ a foaf:Person ;
        foaf:name "Axel Polleres" ;
        rdfs:seeAlso <http://www.polleres.net/foaf.rdf>].
_:me foaf:knows [ a foaf:Person ;
        foaf:name "Wolfgang Faber" ;
        rdfs:seeAlso <http://www.kr.tuwien.ac.at/staff/faber/foaf.rdf>].
_:me foaf:knows [ a foaf:Person ;
        foaf:name "Francesco Calimeri" ;
        rdfs:seeAlso <http://www.mat.unical.it/kali/foaf.rdf>].
_:me foaf:knows [ a foaf:Person .
        foaf:name "Roman Schindlauer" .
        rdfs:seeAlso <http://www.kr.tuwien.ac.at/staff/roman/foaf.rdf>].
```

Fig. 1.1: Giovambattista Ianni's personal FOAF file.

```
_:a foaf:knows _:b .
_:a foaf:knows _:c .
_:b rdf:type foaf:Person .
_:b foaf:name "Bob" .
_:b foaf:knows _:c .
_:c rdf:type foaf:Person .
_:c foaf:name "Charles" .
```

A Turtle shortcut notation like

```
_:a rdf:type foaf:Person ;
    foaf:name "Alice" ;
    foaf:knows _:b ;
    foaf:knows _:c .
```

is a condensed version of the first four triples stated before.

Other common notations for RDF are N-Triples[4] and Notation 3[5].

Figure 1.1 shows some information about one of the authors of this article extracted from RDF data that are available on the Web. RDF defines a special

[4] http://www.w3.org/2001/sw/RDFCore/ntriples/

[5] http://www.w3.org/DesignIssues/Notation3.html

property `rdf:type`,[6] abbreviated in Turtle syntax by the "`a`" letter. It allows the specification of "IS-A" relations, such as, for instance,

```
<http://www.mat.unical.it/~ianni/foaf.rdf> a foaf:PersonalProfileDocument.
```

in Figure 1.1 links the resource `<http://www.mat.unical.it/~ianni/foaf.rdf>` to the resource `foaf:PersonalProfileDocument` via `rdf:type`.

Types supported for RDF property values are URIs, or the two basic types, viz. `rdf:Literal` and `rdf:XMLLiteral`. Under the latter, a basic set of XML schema datatypes are supported.

The RDF Schema (RDFS) is a semantic extension of basic RDF. By giving special meaning to the properties `rdfs:subClassOf` and `rdfs:subPropertyOf`, to `rdfs:domain` and `rdfs:range`, as well as to several types (like `rdfs:Class`, `rdfs:Resource`, `rdfs:Literal`, `rdfs:Datatype`, etc.), RDFS allows to express simple taxonomies and hierarchies among properties and resources, as well as domain and range restrictions for properties.

The semantics of RDFS can be approximated by axioms in FOL, see e.g. [82]. Such a formalization can be used as a basis for RDFS reasoning, where the truth of a given triple t in a given RDF graph G under the RDFS semantics is decided.

1.2.2 The Web Ontology Language OWL

The next layer in the Semantic Web stack serves to formally define domain models as shared conceptualizations, called ontologies [55]. The Web Ontology Language OWL [32] is used to specify such domain models. The W3C document defines three languages OWL Lite, OWL DL, and OWL Full, with increasing expressive power. The first two are syntactic variants of expressive but decidable description logics (DLs) [8]. In particular, OWL DL coincides with with $\mathcal{SHOIN}(\mathbf{D})$ at the cost of imposing several restrictions on the usage of RDFS. 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 RDFS. For in-depth discussion of OWL Full, we refer the interested reader to the language specification [32].

While RDFS itself may already be viewed as a simple ontology language, OWL adds several features beyond RDFS' simple capabilities to define hierarchies (`rdfs:subPropertyOf`, `rdfs:subClassOf`) among properties and classes.[7] In particular, OWL allows to specify transitive, symmetric, functional, inverse, and inverse functional properties. Table 1.1 shows how OWL DL property axioms can be expressed in DL notation and in FOL. RDF triples $S\ P\ O$ are represented here as $P(S, O)$, 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

[6] short for the full URI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`

[7] As conventional in the literature, we use "concept" as a synonym for "class", and "role" as a synonym for "property."

Table 1.1: Mapping 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` $\rangle$	$\top \sqsubseteq \leqslant 1P$	$\forall x, y, z.P(x,y) \wedge P(x,z) \supset y = z$
$\langle P$ `rdf:type owl:InverseFunctionalProperty` $\rangle$	$\top \sqsubseteq \leqslant 1P^-$	$\forall x, y, z.P(x,y) \wedge P(z,y) \supset x = z$
$\langle P$ `rdf:type owl:TransitiveProperty` $\rangle$	$P^+ \sqsubseteq P$	$\forall x, y, z.P(x,y) \wedge P(y,z) \supset P(x,z)$

Table 1.2: 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` $(C_1 \dots C_n)$	$C_1 \sqcap \dots \sqcap C_n$	$C_1(x) \wedge \dots \wedge C_n(x)$
`owl:unionOf` $(C_1 \dots C_n)$	$C_1 \sqcup \dots \sqcup C_n$	$C_1(x) \vee \dots \vee C_n(x)$
`owl:complementOf` (C)	$\neg C$	$\neg C(x)$
`owl:oneOf` $(o_1 \dots o_n)$	$\{o_1, \dots, o_n\}$	$x = o_1 \vee \dots \vee x = o_n$
`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` $(n))$	$\geqslant nP$	$\exists y_1 \dots y_n. \bigwedge_{k=1}^{n} P(x,y_k) \wedge \bigwedge_{i<j} y_i \neq y_j$
`owl:restriction` $(P$ `owl:maxCardinality` $(n))$	$\leqslant nP$	$\forall y_1 \dots y_{n+1}. \bigwedge_{k=1}^{n+1} P(x,y_k) \supset \bigvee_{i<j} y_i = y_j$

*For reasons of legibility, we use a variant of the OWL abstract syntax [91] in this table.

definitions in terms of union or intersection of other classes, as well as restrictions on properties. Table 1.2 gives an overview of the expressive possibilities of OWL for class descriptions and its semantic correspondences with description logics and first-order logics.[8] Such class descriptions can be related to each other using `rdfs:subClassOf`, `owl:equivalentClass`, and `owl:disjointWith` keywords, which allow us to express description logic axioms of the form $C_1 \sqsubseteq C_2$, $C_1 \equiv C_2$, and $C_1 \sqcap C_2 \sqsubseteq \bot$, 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.

For details on the description logic notions used in Tables 1.1 and 1.2, we refer the interested reader to, e.g., [8].

The next, more expressive, iteration of OWL (version 2)[9] is developed by W3C. According to the proposal OWL2 will be based on the decidable description logic $\mathcal{SROIQ}$ [60]. It will support additional features such as acyclic composition

[8] We use a simplified notion for the first-order logic translation here—actually, the translation needs to be applied recursively for any complex DL term. For a formal specification of the correspondence between DL expressions and first-order logic, cf. [8].

[9] `http://www.w3.org/TR/owl2-syntax/`

of properties, qualified number restrictions, and possibility to declare symmetry, reflexivity, or disjointness for properties.

Example 2 (Ontologies in Description Logics). A simple ontology about publications available online at `http://asptut.gibbi.com/sandbox/reviewers.rdf` includes OWL statements which can be represented by the following DL axioms:

$$\exists ex{:}title.\top \sqsubseteq ex{:}Paper \tag{1.1}$$

$$\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string \tag{1.2}$$

$$ex{:}isAuthorOf^{-} \equiv dc{:}creator \tag{1.3}$$

$$ex{:}Publication \equiv ex{:}Paper \sqcap \exists ex{:}publishedIn.\top \tag{1.4}$$

$$\top \sqsubseteq \leqslant 1\ ex{:}publishedIn^{-} \tag{1.5}$$

$$ex{:}Senior \equiv foaf{:}Person \sqcap \geqslant 10\ ex{:}isAuthorOf \sqcap \exists ex{:}isAuthorOf.ex{:}Publication \tag{1.6}$$

The axioms express the following information: *ex*:*title* is a datatype property on *ex*:*Papers* that takes strings as values (axioms (1.1) and (1.2)). Furthermore, the property *ex*:*isAuthorOf* is the inverse of the property *dc*:*creator* (axiom (1.3)). Next, the ontology defines in (1.4) a class *ex*:*Publication* which consists of all the papers which have been published, and in (1.5), we state that *ex*:*publishedIn* to be an inverse functional property (i.e., every paper is published in at most one venue). An *ex*:*Senior* researcher (1.6) is defined as a person who has at least ten papers, some of which are published.

1.2.3 Rule Languages for Integration

The rule languages considered in integration proposals are usually extensions of *Datalog*. Generally, rules have a form of "if" statements, where the predecessor, called the body of the rule, is a Boolean condition and the successor, called the head, specifies a conclusion to be drawn if the condition is satisfied.

In Datalog, the condition of a rule is a conjunction of zero or more atomic formulae of the form $p(t_1, \ldots, t_m)$ where p is an m-ary predicate symbol and $t_1, \ldots, t_m$ are terms which are constant symbols or variables.[10] The head of a rule is an atomic formula (atom). For example, the rule

$$auntOf(X, Y) \leftarrow parentOf(Z, Y), sisterOf(X, Z)$$

states that X is an aunt of Y if Z is a parent of Y and X is this parent's sister. The semantics of Datalog associates with every set of rules (rulebase) its least Herbrand model (see, e.g., [90]), where each ground (i.e., variable-free) atom is associated with a truth value true or false. The least Herbrand model is represented as the set of all atoms assigned to true. These are all the ground

[10] In logic programming, atomic formulae may in addition include terms built with n-ary function symbols.

atoms which follow from the rules interpreted as implications in FOL. For example, the least Herbrand model of the rulebase consisting of the rule above and of the facts $parentOf(tom, john)$, $sisterOf(mary, tom)$ includes the formula $auntOf(mary, john)$. On the other hand, $auntOf(mary, tom)$ does not follow in this rulebase. Datalog with negation uses this to conclude $\neg auntOf(mary, tom)$. Datalog rulebases constitute a subclass of logic programs. The latter use FOL terms, not necessarily restricted to constants and variables. Proposals for integration of rules and ontologies are mostly based on the following extensions of Datalog (which apply also to logic programs):

- **Datalog with negation-as-failure**, where the body may additionally include negation-as-failure (NAF) literals of the form *not a* where a is an atom. Intuitively, a NAF literal *not a* is considered true if it does not follow from the program that a is true. For example, $happy(john)$ can be concluded from the rulebase

$$\begin{aligned} happy(X) &\leftarrow healthy(X), not\ hungry(X) \\ healthy(john) &\leftarrow \end{aligned}$$

  Two commonly accepted formalizations of this intuition are the stable model semantics and the well-founded semantics (see the survey [11]), which are introduced in more detail in Section 1.3. These semantics differ in their view of a belief state as a single classical model in which each atomic fact is either true and false, versus a three-valued model in which each fact is either true, false or *unknown*.
- **Extended Datalog**. This extension (see extended logic programs in [11]) makes it possible to state explicitly negative knowledge. This is achieved by allowing negative literals of the form $-p$, where "$-$" is called the *strong negation* connective, in the heads of rules as well as in the bodies. In addition NAF literals are also allowed in the bodies. For example the rule

$$-healthy(X) \leftarrow hasFever(X)$$

  allows to draw an explicit negative conclusion.
- **Rulebases with priorities**. Datalog rulebases employing strong negation may be inconsistent, i.e., may allow to draw contradictory conclusions. For example, the rules

$$\begin{aligned} fly(X) &\leftarrow bird(X) \\ bird(Y) &\leftarrow penguin(Y) \\ -fly(X) &\leftarrow penguin(X) \\ penguin(tweety) &\leftarrow \end{aligned}$$

  allow to conclude $fly(tweety)$ and $-fly(tweety)$. In Defeasible Logic [2] and in Courteous Logic Programs [54], the user is allowed to specify a priority relation on rules of the rulebase to resolve contradictions in the derived conclusions.

– **Disjunctive Datalog** (see Disjunctive Logic Programs in [11]) admits disjunction of atoms in rule heads, and conjunction of atoms and NAF literals in the bodies, e.g.,

$$male(X) \vee female(X) \leftarrow person(X).$$

A commonly used semantics of Disjunctive Datalog rulebases is an extension of Answer Set Semantics.

The rule languages are supported by implementations which make it possible to query and/or to construct the models of rulebases.

1.2.4 Rule Interchange Format RIF

While there are already standard languages for ontologies viz. RDFS and OWL (which are becoming increasingly used), there is no standard for a rules language available yet.Many rules languages and systems have been proposed, and they offer varying features to reason over Semantic Web data. The *Rule Interchange Format* (RIF) working group of W3C is currently developing a standard exchange format for rules on the Web [13, 12]. The *Rule Interchange Format Basic Logic Dialect* (RIF-BLD) [12] proposed by the group is basically a syntactic variant of Horn rules, which most available rule systems can process.

1.2.5 Approaches to Integration

Integration of a given rule language with a given ontology language is usually achieved by defining a common extension of both, to be called the integrated language. Alternatively, one can adopt an existing knowledge representation language expressive enough to represent rules and ontologies. As OWL is a standard ontology language the ontology languages considered in integration proposals are usually its subsets. The approaches can be classified by the degree of integration of rules and ontologies achieved in the integrated language (see e.g. [4, 81]).

Heterogeneous Integration. In this approach, the distinction between rule predicates and ontology predicates is preserved in the integrated language. Integration of rules and ontologies is achieved by allowing ontology predicates in the rules of the integrated language. Assume for example that an ontology classifies courses as project courses and lecture courses.

$$Project \sqcup Lecture = Course$$

It also includes assertions like $Lecture(cs05)$, $Project(cs21)$ or $Course(cs32)$ (e.g. for courses including lectures and projects). The assertions indicate offered courses. A person is considered a student if he/she is enrolled in an offered lecture or project. This can be expressed by the following rules, using the ontology predicates

$$student(X) \leftarrow enrolled(X, Y), Lecture(Y)$$
$$student(X) \leftarrow enrolled(X, Y), Project(Y)$$

In addition, the rulebase includes enrollment facts, e.g., $enrolled(joe, cs32)$. The extended language allows thus to define ontologies using the constructs of the ontology language and the rulebases with rules referring to the ontologies. An extended rulebase together with an ontology is called a hybrid knowledge base. In heterogeneous approaches, implementations are often based on the hybrid reasoning principle, where a reasoner of the ontology language is interfaced with a reasoner of the rule language to reason in the integrated language.

Two kinds of heterogeneous approaches can be distinguished:

- **Loose coupling.** In this approach, the body of a rule may contain queries to the ontology. A ground set of rules with ontology queries can be reduced to a set of rules without ontology predicates. If the answer to a ground ontology query is positive the query is removed from the rule, otherwise the rule is removed from the set. The semantics of knowledge bases with loose coupling is based on this idea.
  With loose coupling applied to the example above, it cannot be concluded that Joe is a student. This is because neither $Lecture(cs32)$ nor $Project(cs32)$ can be derived from the ontology.
  Examples of loose coupling include:
  - **dl-programs** [46, 43, 47, 42] combining (disjunctive) Datalog with negation under answer set semantics with OWL DL. So-called dl-queries, querying the ontology, are allowed in rule bodies. They may also refer to a variant of the ontology, where the set of its assertions is modified by the dl-query. This enables bi-directional flow of information between rules and ontologies. This work was partly supported by REWERSE and is discussed in more detail in Section 1.3.2.
  - **HEX-programs** [44] extending logic programs under the answer set semantics with support for higher/order and external atoms. This work was partly supported by REWERSE and is discussed in more detail in Section 1.3.3.
  - **TRIPLE** [101] a rule language with the syntax inspired by F-logic which admits queries to the ontology in rule bodies.
  - **SWI Prolog**[11] a logic programming system with a Semantic Web library which makes it possible to invoke RDF Schema and OWL reasoners from Prolog programs.

- **Tight integration.** In this approach, a semantics for the integrated language is given which defines models of hybrid knowledge bases by referring to the semantics of the original rule language and to the FOL models of the ontology. For example, tight integration of Datalog (without negation) with a Description Logic can be achieved within FOL by interpreting Datalog rules as implications. In this semantics, $student(joe)$ is a logical consequence

[11] `http://www.swi-prolog.org/`

of the example hybrid knowledge base. As $Course(cs32)$ is an assertion of the ontology, it follows by the axiom $Project \sqcup Lecture = Course$ that in any FOL model of the ontology $Project(cs32)$ or $Lecture(cs32)$ is true. As $enrolled(joe, cs32)$ is true in every model so the premises of at least one of the implications

$$student(joe) \leftarrow enrolled(joe, cs32), Lecture(cs32)$$
$$student(joe) \leftarrow enrolled(joe, cs32), Project(cs32)$$

must be true in any model. Hence $student(joe)$ is concluded.
Examples of tight integration include:

- $\mathcal{AL}$**-log** [33] and **CARIN** [68], classical works on integrating Datalog with a family of Description Logics under the FOL semantics.
- $\mathcal{DL}$**+log** [97] and its predecessor *r-hybrid knowledge bases* [96] integrating Disjunctive Datalog under Answer Set Semantics with OWL DL. For each FOL model of the ontology, the rules of the knowledge base are reduced to rules of Disjunctive Datalog, with stable models defined by the Answer Set Semantics. Similar to $\mathcal{DL}$+log is the approach of [57]. The guarded hybrid (g-hybrid) knowledge bases introduced therein integrate so-called *guarded programs* with ontologies in a particular DL close to OWL DL.
- **Hybrid Rules** [39] integrating logic programs under the well-founded semantics with OWL DL. For each FOL model of the ontology, the rules of the knowledge base are reduced to a logic program with the model defined by the well-founded semantics. This work was done within REWERSE and is reported in more details in Section 1.3.4.
- **Tightly Coupled dl-Programs** [73] combine disjunctive logic programs under the answer set semantics with description logics. They are based on a well-balanced interface between disjunctive logic programs and description logics, which guarantees the decidability of the resulting formalism without assuming syntactic restrictions. They faithfully extend both disjunctive programs and description logics. We refer to [73] for a detailed comparison to the above loosely coupled dl-programs.

The theoretical foundations developed by studying integration of ontologies with variants of Datalog provide a basis for further extensions. This includes dealing with uncertain and inconsistent knowledge, and using integrated Datalog-based languages as condition languages for ECA-rules.

Homogeneous Integration. The integrated language makes no distinction between rule predicates and ontology predicates. It includes the original rule language and the original ontology language as sublanguages. The integration is to be *faithful* in the sense that the sublanguages should have the same semantics as the respective original languages. Homogeneous integration is difficult to achieve since usually ontology languages are based on FOL and rule languages often support non-monotonic reasoning. An interesting related question is if existing

proposals for heterogeneous integration can be embedded into more expressive logical languages.

Examples of homogeneous integration include:

- **DLP (Description Logic Programs)** [53], a language obtained by intersection of a Description Logic with Datalog rules interpreted as FOL implications. DLP has a limited expressive power, but a DLP ontology can be compiled into rules and easily integrated into a rulebase of a more expressive rule language. For example **Sweet Rules**[12] combine DLP and Datalog with strong negation and priorities. The technique of compiling ontologies to rules is also used in DR-Prolog [3] based on Defeasible Logic [2].
- **SWRL (Semantic Web Rule Language)**[13] extending OWL DL with rules interpreted as FOL implications. Thus SWRL is based on FOL and does not offer nonmonotonic features, such as negation-as-failure. SWRL is undecidable. More recent works define decidable subsets of SWRL: Description Logic Rules [66] and ELP [67].
- **F-logic** [62] extending classical predicate calculus with the concepts of objects, classes, and types. It is expressive enough to represent ontologies, rules and their combinations [61].
- **Hybrid MKNF Knowledge Bases** [85, 87] take Lifschitz's bimodal *Logic of Minimal Knowledge and Negation as Failure (MKNF)* [69] as a basis of faithful integration of Description Logic with Disjunctive Datalog. In addition, more recent results define the well-founded semantics for a subclass of Hybrid MKNF KBs [63, 64].
- **Extended RDF Ontologies** [1] is an extension of RDF graphs with rules which admits NAF and strong negation. A stable model semantics defined for ERDF extends the semantics of RDF Schema. It is based on the partial logic of [56] and supports both closed-world and open-world reasoning.

The issue of embedding existing heterogeneous approaches into unifying logics was addressed by several authors. In particular, [31] shows how the Quantified Equilibrium Logic can be used for embedding heterogeneous approaches, like $\mathcal{DL}$+log and g-hybrid knowledge bases. The first-order autoepistemic logic [65] is considered as a unifying framework for integration of rules and ontologies in [29, 30]. The latter paper shows how dl-programs, r-hybrid knowledge bases and hybrid MKNF knowledge bases can be embedded in this logic.

1.3 Hybrid Rules and Ontologies in REWERSE

In this section, we give a brief exposition of work that has been done in REWERSE regarding the combination of rules and ontologies. In fact, this problem has been approached in different ways, aiming at the support of different semantics and operability of the combination.

[12] `http://sweetrules.projects.semwebcentral.org/`
[13] `http://www.w3.org/Submission/SWRL/`

The main achievements are combinations for the two standard semantics of non-monotonic logic programs to date that were already mentioned in Section 1.2.3, viz. the *stable model semantics* [50] (which is called *answer set semantics* [52] in the version where strong negation is supported), and the *well-founded semantics* [103].

The *stable model semantics* [51] associates with each rulebase some (possibly zero) two-valued Herbrand models called *stable models* (or *answer sets*). Intuitively, a model is stable, if it can be recreated by applying the rules of the program starting from facts, where negation-as-failure in rule bodies is evaluated with respect to that model. Formally, stable models may be defined by using the famous Gelfond-Lifschitz reduct [51].

The *well-founded semantics* [103] instead associates with a rulebase a unique (three-valued) Herbrand model, called the *well-founded* model of P, in which each ground atom is assigned one of three logical values true, false or unknown. Intuitively, the facts of a program should be true, and the ground atoms which are not instances of the head of any rule should be false. This information can be used to reason which other atoms must be true and which must be false in any Herbrand model. Such a reasoning gives in the limit the well-founded model, where the truth values of some atoms may still be undefined.

The properties and relationships between stable and well-founded semantics are well-understood and explored, and we do not embark on this issue here but refer to the literature, cf. [10]. We mention, though, that for a large class of programs relevant in practice (so-called stratified programs [11]), the two semantics coincide.

However, the different nature of the two semantics, and the available methods and algorithms for program evaluation in them is important with respect to possible combinations with ontologies. Indeed, the stable model semantics as a multiple-models semantics has to cope with several possible outcomes (that is, with nondeterminism in the evaluation), while the well-founded semantics as a canonical model semantics is determined; this makes it also more amenable to use proof-oriented methods for evaluation. In line with this, well-founded semantics engines (e.g., XSB) may be top-town oriented, while stable model engines, by current technology, are very much bottom up oriented (e.g., DLV and Smodels).

Within REWERSE, combinations of rules and ontologies have been developed that fall into the heterogeneous integration class described in Section 1.2.5. More in detail, *non-monotonic dl-programs* [46, 43] and the more general HEX-*programs* [44] have been developed in order to have a loose coupling of OWL ontologies with nonmonotonic logic programs under the answer set semantics, while *Hybrid Rules (HD-rules)* have been developed in order to tightly couple OWL ontologies with nonmonotonic logic programs under the well-founded semantics.

The combinations faithfully extend the underlying logic programming semantics, and prototypes have been implemented that build on existing standard reasoning engines for logic programs and OWL ontologies. In fact, they were the first implementations of this kind, giving REWERSE a lead in the realization of expressive non-monotonic combinations of rules and ontologies. An application

within REWERSE was a tool for computing credentials from rule-based policy specifications, based on the engine for HEX-programs.

In the following subsections, we briefly present the two streams of work that have been carried out by the groups in Linköping and Vienna, respectively. For space reasons, we must confine to the essential aspects and conveying the flavor; more details are available in the background publications.

1.3.1 Extensions of Expressive Non-Monotonic Logic Programs by DL-Programs and HEX-Programs

The first stream of work for combining rules and ontologies in REWERSE was directed towards the stable models and answer set semantics, and led to two formalisms: dl-programs and HEX-programs.

The development of dl-programs was motivated by providing an extension to ordinary logic programs that allows one to couple a logic programming engine and description logic reasoner in a meaningful way. However, apart from the usual software engineering problems in coupling heterogeneous systems, the real challenge consisted in a smooth semantic integration, given that logic programs and OWL ontologies are based on rather different semantic grounds which are difficult to bridge (cf. [40]). To overcome this problem, as described in Section 1.2.5 non-monotonic dl-programs foster a loose integration, which takes an interfacing view where the logic program rules and the OWL ontologies can exchange information in terms of extensional data through so called *description-logic atoms* (dl-atoms), which may appear in the logic program. In a nutshell, such atoms can update and query an ontology, i.e., information can flow in both directions of the integrated knowledge bases.

This concept appeared to be quite fruitful and allows an easy definition of the semantics of dl-programs, by generalizing the stable model resp. answer set semantics of ordinary logic programs in a natural way. Furthermore, abstraction of description logic atoms to generic *external atoms* (which is somewhat related to the notion of generalized quantifiers in logic) opened the door to combine ordinary logic programs not only with ontologies, but with (in principle) any kind of external software via an interface at the extensional level. In particular, this facilitates to access and combine data and information in different formats (e.g., in OWL and RDF simultaneously), and to "out-source" parts of computations from the logic program to external functions, which can use tailored and problem-specific methods; the rules in the logic program then serve the role to generate different scenarios (e.g., by making guesses) and constrain solution candidates, for which the results of different computations might be suitably combined.

It turned out that such capabilities were useful for a problem of credential computation in rule-based policy specifications, and that a prototype for this task could be easily built on top of a prototype implementation of HEX-programs.

1.3.2 DL-Programs

Description logic programs (dl-programs), which had been introduced in [46], are a novel type of hybrid knowledge bases combining description logics and logic programs. They form another contribution to the attempt in finding an appropriate formalisms for combined rules and ontologies for the Semantic Web.

Roughly speaking, dl-programs consist of a normal logic program P and a description logic knowledge base (DL-KB) L. The logic program P might contain special devices called dl-atoms. Those dl-atoms may occur in the body of a rule and involve queries to L. Moreover, dl-atoms can specify an input to L before querying it, thus in dl-programs a bidirectional data flow is possible between the description logic component and the logic program.

The way dl-programs interface DL-KBs allows them to act as loosely coupled formalism. This feature brings the advantage of reusing existing logic programming and DL system in order to build an implementation of dl-programs.

In the following, we provide the syntax of dl-programs and an overview of the semantics. An in-detail treatise is given in [43].

Syntax of DL-Programs. Informally, a dl-program $KB = (L, P)$ consists of a description logic knowledge base L and a generalized normal program P, which may contain queries to L. Roughly, such a query asks whether a specific description logic axiom is entailed by L or not.

We first define dl-queries and dl-atoms, which are used to express queries to the description logic knowledge base L. A *dl-query* $Q(\mathbf{t})$ is either

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

A *dl-atom* has the form

$$\mathrm{DL}[S_1 op_1 p_1, \ldots, S_m op_m\ p_m; Q](\mathbf{t})\,, \qquad m \geq 0, \tag{1.7}$$

where each S_i is either a concept or a role, $op_i \in \{\uplus, \cup, \cap\}$, p_i is a unary resp. binary predicate symbol, and $Q(\mathbf{t})$ is a dl-query. We call $p_1, \ldots, p_m$ its *input predicate symbols*. Intuitively, $op_i = \uplus$ (resp., $op_i = \cup$) increases S_i (resp., $\neg S_i$) by the extension of p_i, while $op_i = \cap$ constrains S_i to p_i.

A *classical literal* (or simply literal) l is an atom p or a negated atom $-p$ with a rule predicate symbol (hence not a predicate symbol of L). A *dl-rule* r has the form

$$a \leftarrow b_1, \ldots, b_n, \text{not } b_{n+1}, \ldots, \text{not } b_m, \tag{1.8}$$

where a is a literal and any literal $b_1, \ldots, b_m$ may be a dl-atom. We define $H(r) = a$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_n\}$ and $B^-(r) = \{b_{n+1}, \ldots, b_m\}$. If $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then r is a *fact*. A *dl-program* $KB = (L, P)$ consists of a description logic knowledge base L and a finite set of dl-rules P.

The next example will illustrate main ideas behind the notion of dl-program.

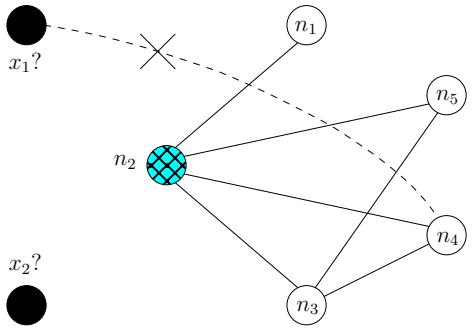


Fig. 1.2: Hightraffic network

Example 3. An existing network must be extended by new nodes (Fig. 1.2). The knowledge base L_N contains information about existing nodes $(n_1, \ldots, n_5)$ and their interconnections as well as a definition of "overloaded" nodes (concept *HighTrafficNode*), which are nodes with more than three connections:

$$\begin{gathered}
\geq 1\ \mathit{wired} \sqsubseteq \mathit{Node};\quad \top \sqsubseteq \forall \mathit{wired}.\mathit{Node};\quad \mathit{wired} = \mathit{wired}^{-};\\
\geq 4\ \mathit{wired} \sqsubseteq \mathit{HighTrafficNode};\quad n_1 \neq n_2 \neq n_3 \neq n_4 \neq n_5;\\
\mathit{Node}(n_1);\quad \mathit{Node}(n_2);\quad \mathit{Node}(n_3);\quad \mathit{Node}(n_4);\quad \mathit{Node}(n_5);\\
\mathit{wired}(n_1, n_2);\quad \mathit{wired}(n_2, n_3);\quad \mathit{wired}(n_2, n_4);\\
\mathit{wired}(n_2, n_5);\quad \mathit{wired}(n_3, n_4);\quad \mathit{wired}(n_3, n_5).
\end{gathered}$$

In L_N, only n_2 is an overloaded node, and is highlighted in Fig. 1.2 with a criss-cross pattern.

To evaluate possible combinations of connecting the new nodes, the following program P_N is specified:

$$\mathit{newnode}(x_1). \tag{1.9}$$

$$\mathit{newnode}(x_2). \tag{1.10}$$

$$\mathit{overloaded}(X) \leftarrow \mathrm{DL}[\mathit{wired} \uplus \mathit{connect}; \mathit{HighTrafficNode}](X). \tag{1.11}$$

$$\begin{aligned}\mathit{connect}(X, Y) \leftarrow\ & \mathit{newnode}(X), \mathrm{DL}[\mathit{Node}](Y),\\ & \text{not } \mathit{overloaded}(Y), \text{not } \mathit{excl}(X, Y).\end{aligned} \tag{1.12}$$

$$\mathit{excl}(X, Y) \leftarrow \mathit{connect}(X, Z), \mathrm{DL}[\mathit{Node}](Y), Y \neq Z. \tag{1.13}$$

$$\mathit{excl}(X, Y) \leftarrow \mathit{connect}(Z, Y), \mathit{newnode}(Z), \mathit{newnode}(X), Z \neq X. \tag{1.14}$$

$$\mathit{excl}(x_1, n_4). \tag{1.15}$$

Rules (1.9)–(1.10) define the new nodes to be added. Rule (1.11) imports knowledge about overloaded nodes in the existing network, taking new connections already into account. Rule (1.12) connects a new node to an existing one, provided the latter is not overloaded and the connection is not to be disallowed, which is specified by Rule (1.13) (there must not be more than one connection

for each new node) and Rule (1.14) (two new nodes cannot be connected to the same existing one). Rule (1.15) states a specific condition: Node x_1 must not be connected with n_4.

Semantics of DL-Programs. Two different semantics have been defined for dl-programs, the (strong) answer-set semantics [46] and the well-founded semantics [47]. The former extends the notion of Gelfond-Lifschitz reduct incorporating the presence of dl-atoms: dl-programs can have, in general, multiple answer sets. The latter extends the well-founded semantics of [103] to dl-programs. The well-founded semantics is based on an appropriate notion of greatest unfounded set which embraces the presence of dl-atoms, and assigns a single three-valued model to every logic program.

More formally, given a consistent set I of classical literals (using the constants in P and L), I *satisfies* (i) a classical ground literal l, denoted $I \models_L l$, iff $l \in I$, and (ii) a dl-atom $a = \mathrm{DL}[\lambda; Q](\mathbf{c})$ with input list $\lambda = S_1 op_1 p_1, \ldots, S_m op_m p_m$, denoted $I \models_L a$, iff $L \cup \lambda(I) \models Q(\mathbf{c})$, where $\lambda(I) = \bigcup_{i=1}^{m} A_i(I)$ and

- $A_i(I) = \{S_i(\mathbf{d}) \mid p_i(\mathbf{d}) \in I\}$, for $op_i = \uplus$;
- $A_i(I) = \{\neg S_i(\mathbf{d}) \mid p_i(\mathbf{d}) \in I\}$, for $op_i = \cup$;
- $A_i(I) = \{\neg S_i(\mathbf{d}) \mid p_i(\mathbf{d}) \in I$ does not hold$\}$, for $op_i = \cap$.

Given a ground dl-rule r, we define (i) $I \models_L B(r)$ iff $I \models_L l$ for all $l \in B^+(r)$ and $I \not\models_L l$ for all $l \in B^-(r)$, and (ii) $I \models_L r$ iff $I \models_L H(r)$ whenever $I \models_L B(r)$. We say that I is a *model* of $KB = (L, P)$, or I *satisfies* KB, denoted $I \models KB$, iff $I \models_L r$ for all r in the grounding of P, $ground(P)$.

Strong answer sets can then be defined as follows. The *Gelfond-Lifschitz transform* of a dl-program $KB = (L, P)$ relative to consistent set I of ground literals for P is the dl-program $KB^I = (L, P^I)$, where P^I is obtained from $ground(P)$ by (i) deleting every rule r with $I \models_L l$ for some $l \in B^-(r)$ (ii) deleting all literals not b_i from all remaining rules. Assuming that all dl-atoms a that occur in P^I are monotone (i.e., $I \models_L a$ implies $I' \models_L a$, for all consistent sets $I \subseteq I'$ of ground literals for P), I is a *strong answer set* of KB iff it is a minimal model (w.r.t. set inclusion) of KB^I. For more details, see [43].

Example 4. As specified by the strong answer set semantics of dl-programs, the program (L_N, P_N) in Example 3 has four strong answer sets (we show only atoms with predicate *connect*): $M_1 = \{connect(x_1, n_1),\ connect(x_2, n_4),\ \ldots\}$, $M_2 = \{connect(x_1, n_1),\ connect(x_2, n_5),\ \ldots\}$, $M_3 = \{connect(x_1, n_5),\ connect(x_2, n_1),\ \ldots\}$, and $M_4 = \{connect(x_1, n_5),\ connect(x_2, n_4),\ \ldots\}$. Note that the ground dl-atom $\mathrm{DL}[wired \uplus connect; HighTrafficNode](n_2)$ from rule (3) is true in any partial interpretation of P_N. According to the proposed well-founded semantics for dl-programs in [47], the unique well-founded model of (L_N, P_N) contains thus $overloaded(n_2)$.

Features and Properties of DL-Programs. The strong answer set semantics of dl-programs is nonmonotonic, and generalizes the stable semantics of ordinary logic programs. In particular, satisfiable positive dl-programs (programs without

default negation and $\sqcap$ operator) have a least model semantics, and satisfiable stratified dl-programs have a unique minimal model which is iteratively described by a finite sequence of least models. Similarly, the well-founded semantics for dl-programs is a generalization of the well-founded semantics for ordinary logic programs. The two generalized semantics preserve some of the relationships that the answer set semantics and the well-founded semantics for normal programs have. In particular, given a knowledge base $KB = (L, P)$, the well-founded model of KB is contained in the set of cautious consequences of KB under strong answer set semantics [47, 42]. Also, the two notions coincide in case stratified programs are considered.

The computational complexity of the formalism does not dramatically increase for dl-programs compared to normal logic programs: under strong answer set semantics, deciding satisfiability of general dl-programs over $\mathcal{SHIF}(\mathbf{D})$ DL-KBs is NEXP-complete, and $\mathrm{P}^{\mathrm{NEXP}}$-complete if the DL-KB is in $\mathcal{SHOIN}(\mathbf{D})$. dl-programs have been generalized to a framework for incorporation of arbitrary knowledge sources other than description logic bases (see Section 1.3.3).

Applications. The bidirectional flow of knowledge between a description logic base and a logic program component enables a variety of possibilities. A major application for dl-programs is nonmonotonic reasoning on top of monotonic systems. We will present two flavors: *default logic* [95] and *closed world assumption* (CWA) [94]. Both reasoning applications can be implemented in dl-programs to support nonmonotonic reasoning for description logics.

We will give an example on how to implement default reasoning on top of ontologies. Since description logics are fragments of first-order logic, Reiter's default logic over description logics can be realized in dl-programs (cf. also terminological default logics [9]).

Let $\Delta = \langle L, D \rangle$ be a default theory, where

$$L = \left\{ \begin{array}{l} redWine \sqsubseteq \neg whiteWine, lambrusco \sqsubseteq sparklingWine \sqcap redWine, \\ sparklingWine(veuveCliquot), lambrusco(lambrusco\_di\_modena) \end{array} \right\}$$

and

$$D = \left\{ \frac{sparklingWine(X) : whiteWine(X)}{whiteWine(X)}, \frac{whiteWine(X) : servedCold(X)}{servedCold(X)} \right\} .$$

The embedding of Δ into a dl-program $KB^{df} = (L, P)$ is demonstrated next. Let P be the program

$$in_{whiteWine}(X) \leftarrow \text{not } out_{whiteWine}(X) \tag{1.16}$$
$$out_{whiteWine}(X) \leftarrow \text{not } in_{whiteWine}(X) \tag{1.17}$$
$$in_{servedCold}(X) \leftarrow \text{not } out_{servedCold}(X) \tag{1.18}$$
$$out_{servedCold}(X) \leftarrow \text{not } in_{servedCold}(X) \tag{1.19}$$
$$fail \leftarrow \mathrm{DL}[\lambda'; whiteWine](X), out_{whiteWine}(X), \text{not } fail \tag{1.20}$$
$$fail \leftarrow \mathrm{DL}[\lambda'; servedCold](X), out_{servedCold}(X), \text{not } fail \tag{1.21}$$

$$g_1(X) \leftarrow \mathrm{DL}[\lambda; \mathit{sparklingWine}](X), \mathrm{not}\, \mathrm{DL}[\lambda'; \neg \mathit{whiteWine}](X) \quad (1.22)$$

$$g_2(X) \leftarrow \mathrm{DL}[\lambda; \mathit{whiteWine}](X), \mathrm{not}\, \mathrm{DL}[\lambda'; \neg \mathit{servedCold}](X) \quad (1.23)$$

$$\mathit{fail} \leftarrow \mathrm{not}\, \mathrm{DL}[\lambda; \mathit{whiteWine}](X), \mathit{in}_{\mathit{whiteWine}}(X), \mathrm{not}\, \mathit{fail} \quad (1.24)$$

$$\mathit{fail} \leftarrow \mathrm{DL}[\lambda; \mathit{whiteWine}](X), \mathit{out}_{\mathit{whiteWine}}(X), \mathrm{not}\, \mathit{fail} \quad (1.25)$$

$$\mathit{fail} \leftarrow \mathrm{not}\, \mathrm{DL}[\lambda; \mathit{servedCold}](X), \mathit{in}_{\mathit{servedCold}}(X), \mathrm{not}\, \mathit{fail} \quad (1.26)$$

$$\mathit{fail} \leftarrow \mathrm{DL}[\lambda; \mathit{servedCold}](X), \mathit{out}_{\mathit{servedCold}}(X), \mathrm{not}\, \mathit{fail} \quad (1.27)$$

corresponding to the default rules in D, where λ and λ' are update lists of form $\mathit{whiteWine} \uplus g_1, \mathit{servedCold} \uplus g_2$ and $\mathit{whiteWine} \uplus \mathit{in}_{\mathit{whiteWine}}, \mathit{servedCold} \uplus \mathit{in}_{\mathit{servedCold}}$, respectively. Intuitively, the predicates $\mathit{in}_{\mathit{whiteWine}}$ and $\mathit{in}_{\mathit{servedCold}}$ encode that an individual is a member of concept $\mathit{whiteWine}$ and $\mathit{servedCold}$, resp. Similarly, $\mathit{out}_{\mathit{whiteWine}}$ and $\mathit{out}_{\mathit{servedCold}}$ are used to state that an individual is not a member of these concepts. The rules (1.16)–(1.19) guess an extension of those predicates, whereas (1.20) and (1.21) check whether the guessed model is compliant with the ontology L. Rules (1.22) and (1.23) are used to test the applicability of the defaults in D, and (1.24)–(1.27) then check whether the guess agrees with the semantics of default logic. In order to have models that agree with the conclusions of L, λ and λ' take over the task to communicate the current world view of P to the ontology L. Under strong answer set semantics, the above program has, as expected, among its cautious consequences the facts $\mathit{in}_{\mathit{whiteWine}}(\mathit{veuveCliquot})$ and $\mathit{out}_{\mathit{whiteWine}}(\mathit{lambrusco})$, which correctly denotes the fact that sparkling wines are white by default. Above encoding has been improved in [28], where various translations from default logic over description logic into cq-programs (see also Section 1.4.3) are given and further analyzed.

A second line of nonmonotonic reasoning is Reiter's CWA [95]. In this reasoning principle, we can infer the negative fact $\neg p(\mathbf{c})$ from a first-order theory T whenever we are unable to prove the positive fact $p(\mathbf{c})$ from T. The CWA of a theory T, denoted $\mathit{CWA}(T)$, is defined as the set of all literals $\{\neg p(\mathbf{c}) \mid T \not\models p(\mathbf{c})\}$.

Take, for instance, the DL knowledge base

$$L = \{\mathit{apple} \sqsubseteq \mathit{fruit}, \mathit{fruit}(\mathit{williams})\}$$

describing that apples are fruits, and that *williams* is a particular fruit. The above knowledge base L leaves open whether *williams* is an apple or not. Closing L by means of CWA enables us to deduce that $\mathit{CWA}(L) \models \neg \mathit{apple}(\mathit{williams})$, i.e., under CWA we can infer that *williams* is not an apple.

A particular encoding of above reasoning task in dl-programs is accomplished by the rule

$$\overline{\mathit{apple}}(X) \leftarrow \mathrm{not}\, \mathrm{DL}[\mathit{apple}](X) \ ,$$

where $\overline{\mathit{apple}}$ is a fresh predicate. Given L, we can now infer $\overline{\mathit{apple}}(\mathit{williams})$.

A well-known drawback of the CWA is that it faces inconsistency in case of disjunctive information. Let $L' = \{\mathit{apple} \sqcup \mathit{pear}(\mathit{williams})\}$, i.e., *williams* is an apple or a pear. Under CWA, we can infer that *williams* is neither an apple nor a pear, which is inconsistent with our assertion in L'. The *extended closed-world assumption* (ECWA) is a refined version of CWA, which is able to treat cases

like the one above in a reasonable manner. Full details on CWA and ECWA in dl-programs is given in [43].

1.3.3 HEX-Programs

HEX-programs [45] are declarative nonmonotonic logic programs with support for external knowledge and higher-order disjunctive rules. In spirit of dl-programs, they allow for a loose coupling between general external knowledge sources and declarative logic programs through the notion of external atoms, which take input from the logic program and exchange inferences with the external source. In addition, meta-reasoning tasks may be accomplished by means of higher-order atoms. HEX-programs are evaluated under a generalized answer-set semantics, thus are in principle capable of capturing many proposed extensions in answer-set programming.

Syntax of HEX-Programs. Let $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ be mutually disjoint sets whose elements are called *constant names*, *variable names*, and *external predicate names*, respectively. Unless explicitly specified, elements from $\mathcal{X}$ (resp., $\mathcal{C}$) are denoted with first letter in upper case (resp., lower case), while elements from $\mathcal{G}$ are prefixed with the "$\&$" symbol. We note that constant names serve both as individual and predicate names.

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 the *arity* of the atom. Intuitively, Y_0 is the predicate name, and we thus also use the more familiar notation $Y_0(Y_1, \ldots, Y_n)$. The atom is *ordinary*, if Y_0 is a constant.

For example, $(x, rdf{:}type, c)$, $node(X)$, and $D(a, b)$, are atoms; the first two are ordinary atoms.

An *external atom* is of the form

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m) \ , \tag{1.28}$$

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two lists of terms (called *input* and *output* lists, respectively), and $\&g \in \mathcal{G}$ is an external predicate name. We assume that $\&g$ has fixed lengths $in(\&g) = n$ and $out(\&g) = m$ for input and output lists, 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: in this respect, an external predicate $\&g$ is equipped with a function $f_{\&g}$ evaluating to true for proper input values.

A *rule* r is of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_m, \mathrm{not}\, \beta_{m+1}, \ldots, \mathrm{not}\, \beta_n \ , \tag{1.29}$$

where $m, k \geq 0$, $\alpha_1, \ldots, \alpha_k$ are atoms, and $\beta_1, \ldots, \beta_n$ are either atoms or external atoms. We define $H(r) = \{\alpha_1, \ldots, \alpha_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ and $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$. If $H(r) = \emptyset$ and $B(r) \neq \emptyset$, then r is a *constraint*, and if $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then r is a *fact*; r is *ordinary*,

if it contains only ordinary atoms. A HEX-*program* is a finite set P of rules. It is *ordinary*, if all rules are ordinary.

We next give an illustrative example.

Example 5 ([44]). Consider the following HEX-program P:

$$subRelation(brotherOf, relativeOf). \tag{1.30}$$
$$brotherOf(john, al). \tag{1.31}$$
$$relativeOf(john, joe). \tag{1.32}$$
$$brotherOf(al, mick). \tag{1.33}$$
$$invites(john, X) \vee skip(X) \leftarrow X \neq john, \&reach[relativeOf, john](X). \tag{1.34}$$
$$R(X, Y) \leftarrow subRelation(P, R), P(X, Y). \tag{1.35}$$
$$someInvited \leftarrow invites(john, X). \tag{1.36}$$
$$\leftarrow \text{not } someInvited. \tag{1.37}$$
$$\leftarrow \&degs[invites](Min, Max), Max > 2. \tag{1.38}$$

Informally, this program randomly selects a certain number of John's relatives for invitation. The first line states that *brotherOf* is a subrelation of *relativeOf*, and the next three lines give concrete facts. The disjunctive rule (1.34) chooses relatives, employing the external predicate *&reach*. This latter predicate takes in input a binary relation e and a node name n, returning the nodes reachable from n when traversing the graph described by e (see the following Example 7). Rule (1.35) axiomatizes subrelation inclusion exploiting higher-order atoms; that is, for those couples of binary predicates p, r for which it holds $subRelation(p, r)$, it must be that $r(x, y)$ holds whenever $p(x, y)$ is true.

The constraints (1.37) and (1.38) ensure that the number of invitees is between 1 and 2, using (for illustration) an external predicate *&degs* from a graph library. Such a predicate has a valuation function $f_{\&degs}$ where $f_{\&degs}(I, e, min, max)$ is true iff *min* and *max* are, respectively, the minimum and maximum vertex degree of the graph induced by the edges contained in the extension of predicate e in interpretation I.

Semantics of HEX-Programs. In the sequel, let P be a HEX-program. The *Herbrand base* of 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, $grnd(r)$, is defined accordingly, and the grounding of program P is given by $grnd(P) = \bigcup_{r \in P} grnd(r)$. Unless specified otherwise, $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ are implicitly given by P.

Example 6 ([44]). Given $\mathcal{C} = \{edge, arc, a, b\}$, ground instances of $E(X, b)$ are for instance $edge(a, b)$, $arc(a, b)$, $a(edge, b)$, and $arc(arc, b)$; ground instances of $\&reach[edge, N](X)$ are all possible combinations where N and X are replaced by elements from $\mathcal{C}$, for instance $\&reach[edge, edge](a)$, $\&reach[edge, arc](b)$, $\&reach[edge, edge](edge)$, 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 $f_{\&g}$ 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 $I \models a$, if and only if $f_{\&g}(I, y_1, \ldots, y_n, x_1, \ldots, x_m) = 1$.

Example 7 ([44]). Let us associate with the external atom *&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 $\&reach[e, b](d)$ since $f_{\&reach}(I, e, b, d) = 1$.

Note that in contrast to the semantics of higher-order atoms, which in essence reduces to first-order logic as customary (cf. [98]), the semantics of external atoms is in spirit of second order logic since it involves predicate extensions.

Considering example 5, as John's relatives are determined to be Al, Joe, and Mick, P has six answer sets, each of which contains one or two of the facts $invites(john, al)$, $invites(john, joe)$, and $invites(john, mick)$.

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 $r \in grnd(P)$. We call P *satisfiable*, if it has some model.

Given a HEX-program P, the *FLP-reduct* of P with respect to $I \subseteq HB_P$, denoted fP^I, is the set of all $r \in grnd(P)$ 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.

In principle, the truth value of an external atom depends on its input and output lists and on the entire model of the program. In practice, however, we can identify certain types of input terms that allow to restrict the input interpretation to specific relations. The Boolean function associated with the external atom $\&reach[edge, a](X)$ for instance will only consider the extension of the predicate *edge* and the constant value a for computing its result, and simply ignore everything else of the given input interpretation.

Features and Properties of HEX-Programs. As mentioned above, HEX-programs are a generalization of dl-programs, consisting indeed in a form of coupling of rules with arbitrary external computation sources, within a declarative logic-based setting. The higher-order features are similar to those of HiLog [26], i.e., the semantics of this high-order extension is still within first-order logic.

The semantics of HEX-programs conservatively extends ordinary answer-set programs, and it is easily extendable to support weak constraints [17]. External predicates can define other ASP features like aggregate functions [48]. Computational complexity of the language depends on external functions. The former is however not affected if external functions evaluate in polynomial time.

The dlvhex prototype,[14] an implementation of HEX-programs, is based on a flexible and modular architecture. The evaluation of the external atoms is realized by plugins, which are loaded at run-time. The pool of available external predicates can be easily customized by third-party developers.

Applications. HEX-programs have been applied in many applications in different contexts. Hoehndorf et al. [59] showed how to combine multiple biomedical upper ontologies by extending the first-order semantics of terminological knowledge with default logic. The corresponding prototype implementation of such kind of system is given by mapping the default rules to HEX-program. Fuzzy extensions of answer-set programs and their relationship to HEX-programs are given in [88, 58]. The former maps fuzzy answer set programs to HEX-programs, whereas the latter defines a fuzzy semantics for HEX-programs and gives a translation to standard HEX-programs. In [89], the planning language $\mathcal{K}^c$ has been introduced which features external function calls in spirit of HEX-programs.

REWERSE has related applications, where a rule-based solution for solving *credential selection problems* (see below) was discussed. We also refer the reader to Chapter 4, which is devoted to reasoning about policies.

As stated in [16], selecting an "appropriate" set of credentials for satisfying trust negotiation tasks is an important problem. Since users typically want to disclose as little sensitive information as possible, an "appropriate" set of credentials is the least sensitive set of credentials needed to obtain a service. This minimization effort is referred to as the *credential selection problem* [16], which will be explained in the following.

In a nutshell, each participant in a rule-based policy specification environment expresses its policies by logic programs, and credentials provided by the requesting client are encoded by facts. The combination of the policies and a set of credentials should satisfy the given authorization request of the client.

More formally, a credential selection problem (CSEL) consists of

- a finite, stratified logic program P, representing the server's and client's policies,
- a goal G modeling the authorization requested by the client,
- a finite set of integrity constraints IC, representing forbidden combinations of credentials,
- a finite set of ground facts C, representing the portfolio of credentials and declarations of the client, and
- a *sensitivity aggregation function* $sen : 2^C \to \Sigma$, where Σ is a finite set (of *sensitivity values*) partially ordered by $\preceq$.

A solution for the credential selection problem is a set $S \subseteq C$ such that

1. $P \cup S \models G$,
2. $P \cup S \cup IC$ is consistent, and
3. $sen(S)$ is minimal among all S which satisfy 1. and 2.

[14] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

Expressing this kind of credential selections is a valuable application for HEX-programs. The next example from [99] shows how to encode a CSEL instance in a HEX-program. The stratified program P is encoded in *Server policy*.

Server policy

```
% if resource is public, no authentication is necessary
allow(download,Resource) :- public(Resource).

% user may download if she has a subscription and is authenticated
allow(download,Resource) :- authenticated(User),
                            hasSubscription(User,Subscription),
                            availableFor(Resource,Subscription).

% user may download if she has paid and is authenticated
allow(download,Resource) :- authenticated(User),
                            paid(User,Resource).

% user is authenticated, if she has a valid credential
authenticated(User) :- valid(Credential),
                       attr(Credential,name,User).

% a selected credential is valid, if its type is trusted
valid(Credential) :- selectedCred(Credential),
                     attr(Credential,type,T),
                     attr(Credential,issuer,CA),
                     isa(T,id),
                     trustedFor(CA,T).

% types that are ids, i.e., a hierarchy of identifiers
isa(id,id).
isa(ssn,id).
isa(passport,id).
isa(driving_license,id).
```

The goal G as fact *resource*(“*paper01234.pdf*”), the set C of the client's credentials, and (implicitly by *credSens*) the set $\Sigma = \{1, 2, 4\}$ of sensitivity values is given below in *Client example*.

Client example

```
hasSubscription("John Doe",law_basic).
hasSubscription("John Doe",computer_basic).

availableFor("paper01234.pdf",computer_basic).

% the client requests this goal G
resource("paper01234.pdf").

% credential authorities and their ID types
trustedFor("Open University",id).
trustedFor("Visa",id).
```

```
trustedFor("UK Government",ssn).

% next are three credentials and their
% associated properties and sensitivities

credential(cr01).
attr(cr01,type,id).
attr(cr01,name,"John Doe").
attr(cr01,issuer,"Open University").
credSens(cr01,1).

credential(cr02).
attr(cr02,type,ssn).
attr(cr02,name,"John Doe").
attr(cr02,issuer,"UK Government").
credSens(cr02,2).

credential(cr03).
attr(cr03,type,id).
attr(cr03,name,"John Doe").
attr(cr03,issuer,"Visa").
credSens(cr03,4).
```

The final part of the CSEL is given below, encoding the minimal $sen(S)$ of credentials $S \subseteq C$, which satisfies the program. The solution is given as ground facts with predicate *sens*. We make use of the external atom *&policy*, whose associated Boolean function $f_{\&policy}(I, p, n) = 1$ iff $n = \sum_{p(c,i) \in I} i$. That is, in a model of the program encoding our CSEL, $polSens(s)$ holds the sum s of sensitivity values i from all ground $sens(c, i)$ atoms.

Optimization rules

```
% open a search space
selectedCred(X) v -selectedCred(X) :- credential(X).

sens(C,S) :- selectedCred(C), credSens(C,S).

% remove models that don't accomplish the goal
:- not allow(download,R), resource(R).

% compute model sensitivity
polSens(S) :- &policy[sens](S).

% select least sensitive model
:~ polSens(S). [S:1]
```

The solution is given in the abridged answer set $\{sens(cr01,1), polSens(1), \ldots\}$, which specifies that credential *cr01* is sufficient for achieving the goal G and is the least sensitive one among the possible subsets of the credentials C. Note that the program makes use of the weak constraint construct `:~ polSens(S). [S:1]`,

whose intuitive meaning is adding a cost s to answer sets in which `polSens(s)` holds, and then selecting optimal answer sets by minimizing the costs (for an in-detail account on weak constraints (in HEX-programs) we refer to [17, 99]).

1.3.4 Extensions of Well-Founded Semantics by Hybrid Well-Founded Semantics

This section gives an introduction to the REWERSE work on tight integration presented in [39, 38]. This work developed a framework for hybrid combination of normal logic programs under the well-founded semantics with various theories of the FOL. The hybrid programs defined in this way extend faithfully both normal programs and the underlying theories. The framework gives principles of implementation showing how a rule engine supporting the well-founded semantics of normal programs can be combined with a reasoner for the underlying theory to get a reasoner for the hybrid programs which is sound w.r.t. their declarative semantics. The implemented instance of the framework was the language of HD-rules [38, 37], integrating Datalog with negation and OWL DL. The framework itself is not restricted to Datalog; compound terms are permitted in addition to constants.

To present this work we first illustrate on an example the well-founded semantics of normal programs. Then we discuss the syntax and the declarative semantics of the hybrid programs. Finally we explain the principles of the operational semantics.

The Well-Founded Semantics. The well-founded semantics associates with a normal logic program a unique *3-valued Herbrand model*, called its *well-founded model*. For the Herbrand base $\mathcal{H}$ of a program denote $\neg\mathcal{H} = \{ \neg a \mid a \in \mathcal{H} \}$. Then a *3-valued Herbrand interpretation* $\mathcal{I}$ of P is a subset of $\mathcal{H} \cup \neg\mathcal{H}$ such that for no ground atom A both A and $\neg A$ are in $\mathcal{I}$. Intuitively, the set $\mathcal{I}$ assigns the truth value **t** (true) to all its members. Thus A is false (has the truth value **f**) in $\mathcal{I}$ iff $\neg A \in \mathcal{I}$, and $\neg A$ is false in $\mathcal{I}$ iff $A \in \mathcal{I}$. If $A \notin \mathcal{I}$ and $\neg A \notin \mathcal{I}$ then the truth value of A (and that of $\neg A$) is **u** (undefined). The truth value of compound formulae is defined in a usual way. For instance the truth value of $F_1 \wedge F_2$ is **t** if the truth values of both F_1 and F_2 are **t**, it is **f** if the truth value of some of them is **f**, and it is **u** if some of them has the truth value **u** and none has **f**. The notation $\mathcal{I} \models_3 F$ will be used to denote that a formula F is true in a 3-valued interpretation $\mathcal{I}$.

We illustrate the notion of the well-founded model by some examples. Several (equivalent) formal definitions can be found elsewhere, see for instance [102, 5, 49].

Example 8. The well-founded model of program $\{ p \leftarrow p;\ q \leftarrow \neg p;\ r \leftarrow q, \neg r \}$ is $\{\neg p, q\}$. Informally, the value of p is false independently from the values of q, r (as p is defined by a single rule $p \leftarrow p$). From $\neg p$ we derive q (by rule $q \leftarrow \neg p$). However neither r nor $\neg r$ can be derived. The program does not have stable models.

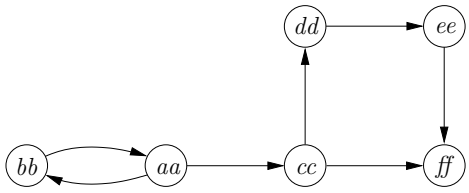


Fig. 1.3: The game graph

Example 9. A two person game consists in moving a token between vertices of a directed graph. Each move consists in traversing one edge from the actual position. Each of the players in order makes one move. The graph is described by a database of facts $m(X, Y)$ corresponding to the edges of the graph. A position X is said to be a *winning position* X if there exists a move from X to a position Y which is a losing (non-winning) position:

$$w(X) \leftarrow m(X, Y), \neg w(Y)$$

Consider the graph in Fig. 1.3 and assume that it is encoded by the facts $m(bb, aa)$, $m(aa, bb), \ldots, m(ee, ff)$ of the program. Now ff is a losing position – there is no move from ff. This is reflected by the well-founded model of the program; in the model $w(ff)$ is false, as no program rule has a ground instance $w(ff) \leftarrow \ldots$ with a true body (as the program contains no fact of the form $m(ff, t)$). Thus ee is a winning position: $w(ee)$ is true, due to rule instance $w(ee) \leftarrow m(ee, ff), \neg w(ff)$. Similarly, $w(cc)$ is true and $w(dd)$ is false. However each of aa, bb is neither winning nor losing, from each of them the player has an option of moving to the other one. Literals $w(aa), w(bb)$ have value **u** in the well-founded model of the program. The model contains the following literals with the predicate symbol w: $w(cc), w(ee), \neg w(dd), \neg w(ff)$.

The program has two stable models, in each of them $w(aa)$ and $w(bb)$ have the opposite logical values (and the values of $w(cc), w(ee), w(dd), w(ff)$ are the same as those in the well-founded model).

The previous sections considered logic programs under the stable model semantics (or, more generally, answer set semantics). Both semantics coincide for a wide class of programs relevant in practice, including the stratified programs. The well-founded model of such a program is 2-valued, and it is its unique stable model. Usually the pragmatics of knowledge representation is different for the two semantics. With the answer set semantics, each stable model represents a solution to a problem. With the well-founded semantics, the solutions are represented by consequences (i.e. answers) of the program.

Hybrid Programs. Informally, a *hybrid program* consists of a set of axioms $\mathcal{T}$, called *external theory* and of a generalized normal program P, which may contain formulae of the language of $\mathcal{T}$, called *constraints*[15] in the bodies of the rules.

[15] This term is used due to similarities with constraint logic programming [83].

More precisely, one considers a first-order alphabet including, as usual, disjoint alphabets of predicate symbols $\mathcal{P}$, function symbols $\mathcal{F}$ (including a set of constants) and variables $\mathcal{V}$. Following the heterogeneous approach to integration, it is assumed that $\mathcal{P}$ consists of two disjoint sets $\mathcal{P}_R$ (*rule predicates*) and $\mathcal{P}_C$ (*constraint predicates*). The atoms and the literals constructed with these predicates are called respectively *rule atoms* (*rule literals*) and *constraint atoms* (*constraint literals*). The bodies of the rules of normal programs over alphabets $\mathcal{P}_R$, $\mathcal{F}$, $\mathcal{V}$ may now be extended with constraints over alphabets $\mathcal{P}_C$, $\mathcal{F}$, $\mathcal{V}$. (It is also allowed that the set of function symbols of the external theory $\mathcal{T}$ is a subset of $\mathcal{F}$.) In a particular instance of the framework one has to be specific about the kind of formulae allowed as constraints of the rules.

A *hybrid rule* has the form

$$a \leftarrow c, b_1, \ldots, b_n, \tag{1.39}$$

where c is a constraint over $\mathcal{P}_C$, $\mathcal{F}$, $\mathcal{V}$ and $b_1, \ldots, b_n$ are rule literals. If constraints allow quantifiers, some variables may not be free in a rule. A safeness restriction on the syntax of rules introduced in [36] for discussing semantic issues is somewhat elaborate. A sufficient condition for a rule to be safe is that each its free variable has to appear in a positive rule literal.

A *hybrid program* is a pair $(P, \mathcal{T})$ where P is a set of hybrid rules and $\mathcal{T}$ is a set of axioms over $\mathcal{P}_C$, $\mathcal{F}$, $\mathcal{V}$. A hybrid program is said to be safe if all its rules are safe.

Remember that we deal with two kinds of negation: the classical negation of FOL and nonmonotonic negation of the rules. The former is applied to (formulae containing only) constraint predicates, and the latter only to (atoms with) rule predicates. So the same symbol $\neg$ can be used to denote both.

The following example [39] shows a safe hybrid program with constraints referring to an ontology.

Example 10. Consider a classification of geographical locations. For example the classification may concern the country (Finland (Fi), Norway (No), etc.), the continent (Europe (E), etc.), and possibly other categories. We specify a classification by axioms in a DL logic. The ontology provides, among others, the following information

– subclass relations (T-box axioms): e.g. ($Fi \sqsubseteq E$);
– classification of some given locations represented by constants (A-box axioms). For instance, assuming that the positions of Example 9 represent locations we may have: bb is a location in Finland ($Fi(bb)$), cc is a location in Europe ($E(cc)$).

Now the ontology will be used as an external theory for a program. We describe a variant of the game from Example 9, with the rules subject to additional restrictions (see Fig. 1.4). Assume that the positions of the graph represent geographical locations described by the ontology. The restrictions will be expressed as ontological constraints added in rule bodies. For instance let constraints be added to the facts $m(ee, ff)$ and $m(cc, ff)$:

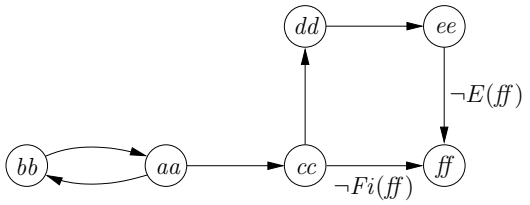


Fig. 1.4: The modified game graph

$$w(X) \leftarrow m(X,Y), \neg w(Y)$$

$$\begin{array}{ll} m(bb,aa) & m(cc,ff) \leftarrow \neg Fi(ff) \\ m(aa,bb) & m(ee,ff) \leftarrow E(ff) \\ m(aa,cc) & \\ m(cc,dd) & \\ m(dd,ee) & \end{array}$$

Intuitively, this would mean that the move from *ee* to *ff* is allowed only if *ff* is in Europe and the move from *cc* to *ff* – only if *ff* is not in Finland. These restrictions may influence the outcome of the game: *ff* will still be a losing position but if the axioms of the ontology do not allow to conclude that *ff* is in Europe, we cannot conclude that *ee* is a winning position. However, we can conclude that if *ff* is not in Europe then it cannot be in Finland. Thus, at least one of the conditions $E(ff), \neg Fi(ff)$ holds. Therefore *cc* is a winning position: If $E(ff)$ then, as in Example 9, *ee* is a winning position, *dd* is a losing one, hence *cc* is a winning position. On the other hand, if $\neg Fi(ff)$ the move from *cc* to *ff* is allowed in which case *cc* is a winning position.

An example employing non-nullary function symbols is given in [37].

Declarative Semantics. The declarative semantics of hybrid programs is defined as a generalization of the well-founded semantics of normal programs; it refers to the (2-valued) models of the external theory $\mathcal{T}$ of a hybrid program. Given a hybrid program $(P, \mathcal{T})$ we cannot define a unique well-founded model of P since we have to take into consideration the logical values of the constraints in the rules. However, for any given model M of $\mathcal{T}$ one can consider the well-founded model of the normal program P/M obtained by replacing the constraints in the rules by their logical values in M.

More precisely, let $ground(P)$ be the set of ground instances of the hybrid rules in P. Then P/M is the normal program obtained from $ground(P)$ by

- removing each rule constraint C which is true in M (i.e. $M \models C$),
- removing each rule whose constraint C is not true in M, (i.e. $M \not\models C$).

The well-founded model of P/M is called the *well-founded model of P based on M*.

A formula F (over $\mathcal{P}_R, \mathcal{F}, \mathcal{V}$) **holds** (is *true*) in the well-founded semantics of a hybrid program $(P, \mathcal{T})$ (denoted $(P, \mathcal{T}) \models_{\text{wf}} F$) iff $M \models_3 F$ for each well-founded model M of $(P, \mathcal{T})$.

We say that F is *false* in the well-founded semantics of $(P, \mathcal{T})$ if $(P, \mathcal{T}) \models_{\text{wf}} \neg F$, and that F is *undefined* if the logical value of F in each well-founded model of $(P, \mathcal{T})$ is **u**. Notice that there is a fourth case: if F does not have the same logical value in all well-founded models of P then F is neither true, nor false, nor undefined. Notice that the negation in the rule literals is nonmonotonic, and the negation in the constraints is that from the external theory, thus monotonic.

Example 11. For the hybrid program $(P, \mathcal{T})$ of Example 10 we have to consider models of the ontology $\mathcal{T}$. For every model M_0 of $\mathcal{T}$ such that $M_0 \models E(\mathit{ff})$ the program P/M_0 includes the fact $m(\mathit{ee}, \mathit{ff})$. The well-founded model of P/M_0 includes thus the literals $\neg w(\mathit{ff}), w(\mathit{ee}), \neg w(\mathit{dd}), w(\mathit{cc})$ (independently of whether $M_0 \models Fi(\mathit{ff})$).

On the other hand, for every model M_1 of the ontology such that $M_1 \models \neg Fi(\mathit{ff})$ the program P/M_1 includes the fact $m(\mathit{cc}, \mathit{ff})$. The well-founded model of P/M_1 includes thus the literals $\neg w(\mathit{ff}), w(\mathit{cc})$ (independently of whether $M_1 \models E(\mathit{ff})$).

Notice that each of the models of the ontology falls in one of the above discussed cases. Thus, $w(\mathit{cc})$ and $\neg w(\mathit{ff})$ hold in the well-founded semantics of the hybrid program, while $w(\mathit{ee}), \neg w(\mathit{ee}), w(\mathit{dd})$ and $\neg w(\mathit{dd})$ do not hold in it (provided that the logical value of $E(\mathit{ff})$ is not the same in all the models of $\mathcal{T}$). The logical value of $w(\mathit{aa})$ and that of $w(\mathit{bb})$ is **u** in each well-founded model of the program. Thus $w(\mathit{aa})$ and $w(\mathit{bb})$ are undefined in the well-founded semantics of the program, and $w(\mathit{dd})$ and $w(\mathit{ee})$ are not (they are neither true, nor false, nor undefined).

Consider a case of hybrid rules without negative rule literals. So the nonmonotonic negation does not occur. Such rules can be seen as implications of FOL and treated as axioms added to $\mathcal{T}$. For such case the well-founded semantics and the logical consequence $\models$ of FOL are similar. They are not equivalent, as the well-founded semantics deals only with Herbrand models of the rules. However they coincide in the following sense. (1) For any ground rule atom A if $(P, \mathcal{T}) \models_{\text{wf}} A$ then $P \cup \mathcal{T} \models A$. The reverse implication does not hold[16]. (2) Assume that only such interpretation domains are considered in which each element is a value of a ground term, and the values of distinct terms are distinct. Then A is true in all models of $P \cup \mathcal{T}$ iff $(P, \mathcal{T}) \models_{\text{wf}} A$, for any rule atom A.

As the well-founded semantics of normal programs is undecidable, so is the well-founded semantics of hybrid programs. It is however decidable for Datalog hybrid programs with decidable external theories.

The declarative semantics of hybrid programs is based on Herbrand models of the rules. Thus it treats distinct terms as having distinct values. The syntactic

[16] As a counterexample take $P = \{\, p \leftarrow q(x), r(x);\ r(x) \leftarrow \,\}$ and $\mathcal{T} = \{\, \exists x.q(x) \,\}$. $P \cup \mathcal{T} \models p$ but $(P, \mathcal{T}) \not\models_{\text{wf}} p$, as there exist models of $\mathcal{T}$ in which each ground atom $q(t)$ is false.

equality of the well-founded semantics may be different from the equality of the external theory. This may lead to strange consequences. For instance consider a hybrid program $(P, \mathcal{T})$, where $P = \{ p(a) \}$. Both $p(a)$ and $\neg p(b)$ hold in the well-founded semantics of $(P, \mathcal{T})$, even if $\mathcal{T}$ implies that $a = b$. One may avoid such anomalies by requiring that – speaking informally – terms which are equal according to $\mathcal{T}$ are treated in the same way by the rules of P. For more details the reader is referred to [36]. To avoid technical difficulties let us require that, in what follows, the external theory satisfies the axioms of the free equality theory (CET, Clark equality theory [27]). Thus ground terms have the same values (in a model of $\mathcal{T}$) iff they are syntactically equal.

Operational Semantics. In this section we present the operational semantics [39, 36] of hybrid programs. The semantics is a basis for implementation; a prototype implementation has been described in [38, 37]. Our presentation is informal. For a precise description the user is referred to [39, 36].

Like in logic programming, the task of a computation is to find instances of a given goal formula G which are true in the well-founded semantics of a given program. Similarly to logic programming, the operational semantics is defined in terms of search trees. It is based on the idea of constructive negation presented in [34, 35]. In that work the only constraint predicate was the equality and the constraint theory was the free equality theory (CET) [27].

The operational semantics is similar to SLDNF- and SLS-resolution [70, 93], extended by handling constraints originating from the hybrid rules. For an input goal a derivation tree is constructed; its nodes are goals. Whenever a negative literal is selected in some node, a subsidiary derivation tree is constructed. In logic programming an answer to a goal is a binding for goal variables. In hybrid programs an answer is a constraint satisfiable in a given theory $\mathcal{T}$. Thus, to develop an implementation it is necessary to have a constraint solver for $\mathcal{T}$. However, the constraint solver is only used as a black box deciding the satisfiability of a given constraint.

A *hybrid goal* (shortly: *goal*) has the form

$$c, b_1, \ldots, b_m$$

where $m \geq 0$, each b_i is a rule literal and c is a constraint, called the constraint of the goal. The definition of safeness and the sufficient condition for safeness applies also to hybrid goals.

The computation is controlled by a selection function which selects a rule literal in a goal. If the selected literal is positive the goal is resolved, as usual, by matching the selected rule literal with the head of a renamed variant of a hybrid rule of the program. However, the unification is replaced by adding a constraint to the derived goal. More precisely, consider a goal $G = c, \overline{L}, b, \overline{L'}$ and a rule $r = h \leftarrow c', \overline{K}$, such that no variable occurs both in G and r. The goal

$$G' \;=\; b{=}h, c, c', \overline{L}, \overline{K}, \overline{L'}$$

is said to be **derived** from G by r, with the selected atom b, if the constraint $b{=}h, c, c'$ is satisfiable. As usual, several rules of the program may match the

selected atom and give rise to different computations, visualized by a tree with nodes labeled by goals and edges representing the derivation steps. A goal c with no rule literals is called *successful* (and ends a *successful branch* of the tree).

Consider such a tree with root G, with no negative literal selected. Let us denote by $c|_G$ the constraint $\exists \ldots c$, where the quantified variables are those variables of c that do not occur (free) in G. Then by an *answer* of the tree we mean any constraint $(c_1 \vee \ldots \vee c_n)|_G$, where $c_1, \ldots, c_n$ are some of the successful leaves of the tree. Every answer, speaking informally, implies G in the well-founded semantics of the program. (A precise formulation is given later on.) The most general answer is the disjunction of all the successful leaves. If the Herbrand universe is infinite, the set of (the constraints of) successful leaves may be infinite and the most general answers may not exist.

The negation of the most general answer is a *negative answer* of the tree; it implies $\neg G$ in the well-founded semantics of the program. Less general negative answers may be obtained without constructing the whole tree. By a *cross-section* we mean a set F of tree nodes such that each successful branch has a node in F. If $c_1, \ldots, c_n$ are the constraints of the goals of a cross-section then $\neg((c_1 \vee \ldots \vee c_n)|_G)$ is a negative answer. We skip here a definition of negative answers corresponding to infinite cross-sections. Each negative answer implies that the root G is false.

In the general case negative literals may be selected in the tree, and we have to deal with three logical values **t**, **u**, **f**. Due to this we introduce two kinds of trees, *t-trees* and *tu-trees*. A t-tree tells when its root G is **t** (in the well-founded semantics of the program). Speaking informally, each answer of the t-tree implies G. A tu-tree tells when its root G is **t** or **u**; G being **t** or **u** implies some answer of the tree. Thus each negative answer of the tu-tree implies $\neg G$. We are interested in the answers of t-trees and the negative answers of tu-trees.

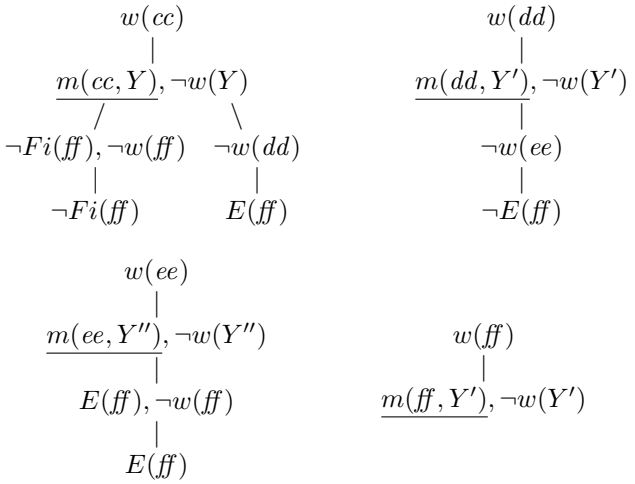
The two kinds of trees differ by the treatment of negative selected literals. In a t-tree, when a negative literal $\neg b$ is selected in a goal $G' = c, \overline{L}, \neg b, \overline{L'}$ then a subsidiary tu-tree for c, b is constructed, and some its negative answer d is obtained. The literal $\neg b$ is replaced by d. If the resulting constraint c, d is satisfiable then the obtained goal $G'' = c, d, \overline{L}, \overline{L'}$ is the (only) child of G' in the t-tree. Otherwise (c, d unsatisfiable) G is a leaf. (An informal justification is that d implies that $\neg b$ is **t**.) One may avoid constructing the subsidiary tu-tree; then G' does not have a child (as $d = \neg c$ is a trivial negative answer of any tu-tree for c, b, and $c, \neg c$ is unsatisfiable).

In a tu-tree, when a negative literal $\neg b$ is selected in a goal $G' = c, \overline{L}, \neg b, \overline{L'}$ then a subsidiary t-tree for c, b is constructed, and some its answer c' is obtained. The literal $\neg b$ is replaced by the negation $d = \neg c'$ of c'. If c, d is satisfiable then the obtained goal $G'' = c, d, \overline{L}, \overline{L'}$ is the (only) child of G' in the tu-tree. Otherwise G' is a leaf. (An informal justification is that c' implies $\neg b$ being **f**; hence $\neg b$ being **t** or **u** implies d.) One may avoid constructing the subsidiary tu-tree; then G' has $G'' = c, \overline{L}, \overline{L'}$ as its child (as $c' =$ **false** is a trivial answer of any t-tree).

To avoid circularity (e.g. a t-tree for p refers to a tu-tree for q, and the latter tree refers to the former) *ranks* are assigned to the trees, similarly as it is done in the definitions of SLDNF- and SLS-resolutions [70, 93].

We now illustrate the operational semantics of hybrid programs by an example. In the example we apply certain simplifications to the tree nodes.[17] The same example without the simplifications is presented in [39, 36].

Example 12. Consider the hybrid program of Example 10. A query $w(cc)$ can be answered by constructing the following trees: a t-tree for $w(cc)$, a tu-tree for $w(dd)$, a t-tree for $w(ee)$, and a tu-tree for $w(\mathit{ff})$; of ranks $3, 2, 1, 0$ respectively. In the goals with more than one rule literals, the selected one is underscored.



The empty cross-section of the tu-tree for $w(\mathit{ff})$ provides a negative answer **true**, the t-tree for $w(ee)$ has an answer $E(\mathit{ff})$, the tu-tree for $w(dd)$ has a negative answer $E(\mathit{ff})$ (the cross-section consisting of the leaf), and the t-tree for $w(cc)$ has an answer $\neg Fi(\mathit{ff}) \vee E(\mathit{ff})$ (which in $\mathcal{T}$ is equivalent to **true**).

An implementation of hybrid programs based on the described ideas is presented in [38, 37]. The operational semantics makes it possible to employ an existing constraint solver (e.g. a description logic reasoner) and treat it as a black box. Also, construction of t-trees and tu-trees can be implemented on top of a Prolog system with the well-founded semantics. Thus the costs of implementation is rather low. We also mention that – similarly as in CLP – it is not necessary to check satisfiability of the constraint for each tree node. The answers (negative answers) of trees obtained in such way are logically equivalent to those described

[17] Any constraint may be replaced by an equivalent one. In any node $C, \overline{L}$ of a t- or tu- tree for G, the constraint C of the node can be replaced by $C|_{G,\overline{L}}$. Instead of referring to a lower rank tree for C, A, a tree for $(C|_A), A$ can be used. Also, a goal may be replaced by a logically equivalent one (e.g. $X{=}a, p(X)$ by $X{=}a, p(a)$). These modifications do not change the (negative) answers of the trees. This is rather obvious in the particular example; we omit a formal justification for a general case.

above. This provides an opportunity to decrease the interaction with the constraint solver, and to improve efficiency. The prototype of [38, 37] invokes the solver once, after having constructed the main tree.

For safe hybrid programs the operational semantics is sound w.r.t. to the well-founded semantics. More precisely if $(P, \mathcal{T})$ is a safe hybrid program and $G = c_0, \overline{L}$ a goal then for any substitution θ

1. if c is an answer of a t-tree for $(P, \mathcal{T})$ and G, and $\mathcal{T} \models c\theta$ then $(P, \mathcal{T}) \models_{\text{wf}} \overline{L}\theta$;
2. if c is a negative answer of a tu-tree for $(P, \mathcal{T})$ and G, and $\mathcal{T} \models c\theta$ then $(P, \mathcal{T}) \models_{\text{wf}} \neg\overline{L}\theta$.

The safeness condition may be abandoned, if additional restrictions are imposed on the existential quantifier used in constraints see [36] for details). This is related to the fact that constraints are interpreted on arbitrary domains, without assuming that each element of a domain is represented by ground term, while the well-founded semantics defines a Herbrand model.

In the general case, the operational semantics is not complete. The reason is that only finite constraint formulae are used as (negative) answers. However the method is complete in the case of Datalog, with safe rules and goals. More precisely, assume that the Herbrand universe is finite. Consider a safe program $(P, \mathcal{T})$ and a safe goal $G = c_0, \overline{L}$. For any grounding substitution θ for the variables of G such that $c_0\theta$ is satisfiable

1. if $(P, \mathcal{T}) \models_{\text{wf}} \overline{L}\theta$ then there exists a t-tree (of a finite rank) for G with an answer c such that $\mathcal{T} \models c\theta$;
2. if $(P, \mathcal{T}) \models_{\text{wf}} \neg\overline{L}\theta$ then there exists a tu-tree (of a finite rank) for G with a negative answer c such that $\mathcal{T} \models c\theta$.

A stronger result, which includes independence from the selection rule, also holds.

1.4 Variants and Extensions of the Basic Formalisms

In this section, we discuss some variants and extensions of the above basic formalisms, which have been crafted in order to make them more versatile or to overcome some restrictions. More specifically, we summarize extensions of the basic formalisms that allow for handling uncertainty and vagueness. We also describe an extension of loosely coupled dl-programs by (unions of) conjunctive queries as dl-atoms and disjunctions in rule heads (called *cq-programs*).

From a more general perspective, during the recent years, handling uncertainty and vagueness has started to play an important role in Semantic Web research. A recent forum for approaches to uncertainty reasoning in the Semantic Web is the annual *Workshop on Uncertainty Reasoning for the Semantic Web (URSW).* There also exists a W3C Incubator Group on *Uncertainty Reasoning for the World Wide Web.* The research focuses especially on probabilistic and fuzzy extensions of description logics, ontology languages, and formalisms integrating rules and ontologies. Note that probabilistic formalisms allow to encode ambiguous information, such as "John is a student with the probability 0.7 and a teacher with

the probability 0.3", while fuzzy approaches allow to encode vague or imprecise information, such as "John is tall with the degree of truth 0.7". Formalisms for dealing with uncertainty and vagueness are especially applied in ontology mapping, data integration, information retrieval, and database querying. Vagueness and imprecision also abound in multimedia information processing and retrieval, and are an important aspect of natural language interfaces to the Web.

We first consider extensions of dl-programs by probabilistic uncertainty, and we then discuss fuzzy extensions. We finally focus on cq-programs.

1.4.1 Probabilistic DL-Programs

We now summarize the main ideas behind loosely and tightly coupled probabilistic dl-programs, introduced in [71, 74, 75, 19] and [18, 22, 20, 21], respectively. For further details on the syntax and semantics of these programs, their background, and their semantic and computational properties, we refer to the above works.

Loosely coupled probabilistic dl-programs [71, 74, 75] are a combination of loosely coupled dl-programs under the answer set and the well-founded semantics with probabilistic uncertainty as in Bayesian networks. Roughly, they consist of a loosely coupled dl-program (L, P) under different "total choices" B (they are the full joint instantiations of a set of random variables, and they serve as pairwise exclusive and exhaustive possible worlds), and a probability distribution μ over the set of total choices B. One then obtains a probability distribution over Herbrand models, since every total choice B along with the loosely coupled dl-program produces a set of Herbrand models of which the probabilities sum up to $\mu(B)$. As in the classical case, the answer set semantics of loosely coupled probabilistic dl-programs is a refinement of the well-founded semantics of loosely coupled probabilistic dl-programs. Consistency checking and tight query processing (i.e., computing the entailed tight interval for the probability of a conditional or unconditional event) for in such probabilistic dl-programs under the answer set semantics can be reduced to consistency checking and query processing in loosely coupled dl-programs under the answer set semantics, while tight query processing under the well-founded semantics can be done in an anytime fashion by reduction to loosely coupled dl-programs under the well-founded semantics. For suitably restricted description logic components, the latter can be done in polynomial time in the data complexity. Query processing in the special case of stratified loosely coupled probabilistic dl-programs can be reduced to computing the canonical model of stratified loosely coupled dl-programs. Loosely coupled probabilistic dl-programs can especially be used for (database-oriented) probabilistic data integration in the Semantic Web, where probabilistic uncertainty is used to handle inconsistencies between different data sources [19].

Example 13. A university database may use a loosely coupled dl-program (L, P) to encode ontological and rule-based knowledge about students and exams. A probabilistic dl-program $KB = (L, P', C, \mu)$ then additionally allows for encoding probabilistic knowledge. For example, the following two probabilistic rules in P' along with a probability distribution on a set of random variables may express

that if two master (resp., bachelor) students have given the same exam, then there is a probability of 0.9 (resp., 0.7) that they are friends:

$$\begin{aligned} \mathit{friends}(X,Y) \;\leftarrow\; & \mathit{given\_same\_exam}(X,Y), DL[\mathit{master\_student}(X)], \\ & DL[\mathit{master\_student}(Y)],\ \mathit{choice}_m\,; \\ \mathit{friends}(X,Y) \;\leftarrow\; & \mathit{given\_same\_exam}(X,Y), DL[\mathit{bachelor\_student}(X)], \\ & DL[\mathit{bachelor\_student}(Y)],\ \mathit{choice}_b\,. \end{aligned}$$

Here, we assume the set $C = \{\{\mathit{choice}_m, \mathit{not\_choice}_m\}, \{\mathit{choice}_b, \mathit{not\_choice}_b\}\}$ of values of two random variables and the probability distribution μ on all their four joint instantiations, given by μ: $\mathit{choice}_m, \mathit{not\_choice}_m, \mathit{choice}_b, \mathit{not\_choice}_b \mapsto 0.9, 0.1, 0.7, 0.3$ under probabilistic independence. For example, choice_m, choice_b is associated with the probability $0.9 \times 0.7 = 0.63$. Asking about the entailed tight interval for the probability that *john* and *bill* are friends can then be expressed by a probabilistic query of the form $\exists(\mathit{friends}(\mathit{john}, \mathit{bill}))[R, S]$, whose answer depends on the available concrete knowledge about *john* and *bill* (whether they have given the same exams, and are both master or bachelor students).

Tightly coupled probabilistic dl-programs [18, 22] are a tight combination of disjunctive logic programs under the answer set semantics with description logics and Bayesian probabilities. They are a logic-based representation formalism that naturally fits into the landscape of Semantic Web languages. Tightly coupled probabilistic dl-programs can especially be used for representing mappings between ontologies [20, 21], which are a common way of approaching the semantic heterogeneity problem on the Semantic Web. In this application, they allow in particular for resolving inconsistencies and for merging mappings from different matchers based on the level of confidence assigned to different rules (see below). Furthermore, tightly coupled probabilistic description logic programs also provide a natural integration of ontologies, action languages, and Bayesian probabilities towards Web Services. Consistency checking and query processing in tightly coupled probabilistic dl-programs can be reduced to consistency checking and cautious/brave reasoning, respectively, in tightly coupled disjunctive dl-programs. Under certain restrictions, these problems have a polynomial data complexity.

Example 14. The two correspondences between two ontologies O_1 and O_2 that (i) an element of *Collection* in O_1 is an element of *Book* in O_2 with the probability 0.62, and (ii) an element of *Proceedings* in O_1 is an element of *Proceedings* in O_2 with the probability 0.73 (found by the matching system hmatch) can be expressed by the following two probabilistic rules:

$$\begin{aligned} & O_2 : \mathit{Book}(X) \leftarrow O_1 : \mathit{Collection}(X) \wedge \mathit{hmatch}_1; \\ & O_2 : \mathit{Proceedings}(X) \leftarrow O_1 : \mathit{Proceedings}(X) \wedge \mathit{hmatch}_2. \end{aligned}$$

Here, we assume the set $\mathcal{C} = \{\{\mathit{hmatch}_i, \mathit{not\_hmatch}_i\} \mid i \in \{1, 2\}\}$ of values of random variables and the probability distribution μ on all joint instantiations of these variables, given by μ: $\mathit{hmatch}_1, \mathit{not\_hmatch}_1, \mathit{hmatch}_2, \mathit{not\_hmatch}_2 \mapsto 0.62, 0.38, 0.73, 0.27$ under probabilistic independence.

Similarly, two other correspondences between O_1 and O_2 (found by the matching system falcon) are expressed by the following two probabilistic rules:

$$O_2 : InCollection(X) \leftarrow O_1 : Collection(X) \wedge falcon_1;$$
$$O_2 : Proceedings(X) \leftarrow O_1 : Proceedings(X) \wedge falcon_2,$$

where we assume the set $\mathcal{C}' = \{\{falcon_i, not\_falcon_i\} \mid i \in \{1, 2\}\}$ of values of random variables and the probability distribution μ' on all joint instantiations of these variables, given by $\mu' : falcon_1, not\_falcon_1, falcon_2, not\_falcon_2 \mapsto 0.94, 0.06, 0.96, 0.04$ under probabilistic independence.

Using the trust probabilities 0.55 and 0.45 for hmatch and falcon, respectively, for resolving inconsistencies between rules, we can now define a merged mapping set that consists of the following probabilistic rules:

$$O_2 : Book(X) \leftarrow O_1 : Collection(X) \wedge hmatch_1 \wedge sel\_hmatch_1;$$
$$O_2 : InCollection(X) \leftarrow O_1 : Collection(X) \wedge falcon_1 \wedge sel\_falcon_1;$$
$$O_2 : Proceedings(X) \leftarrow O_1 : Proceedings(X) \wedge hmatch_2;$$
$$O_2 : Proceedings(X) \leftarrow O_1 : Proceedings(X) \wedge falcon_2,$$

Here, we assume the set $\mathcal{C}''$ of values of random variables and the probability distribution μ'' on all joint instantiations of these variables, which are obtained from $\mathcal{C} \cup \mathcal{C}'$ and $\mu \cdot \mu'$ (which is defined as $(\mu \cdot \mu')(B\, B') = \mu(B) \cdot \mu'(B')$, for all joint instantiations B of $\mathcal{C}$ and B' of $\mathcal{C}'$), respectively, by adding the values $\{sel\_hmatch_1, sel\_falcon_1\}$ of a new random variable along with the probabilities $sel\_hmatch_1, sel\_falcon_1 \mapsto 0.55, 0.45$ under probabilistic independence, for resolving the inconsistency between the first two rules.

1.4.2 Fuzzy DL-Programs

We next briefly describe loosely and tightly coupled fuzzy dl-programs, which have been introduced in [72, 76] and [78, 80], respectively, and extended by probabilities in [77] and by a top-k retrieval technique in [79], respectively. All these fuzzy dl-programs have natural special cases where query processing can be done in polynomial time in the data complexity. For further details on their syntax and semantics, background, and properties, we refer to the above works.

Towards dealing with vagueness and imprecision in the reasoning layers of the Semantic Web, loosely coupled (normal) fuzzy dl-programs under the answer set semantics [72, 76] are a generalization of normal dl-programs under the answer set semantics by fuzzy vagueness and imprecision in both the description logic and the logic program component. This is the first approach to fuzzy dl-programs that may contain default negations in rule bodies. Query processing in such fuzzy dl-programs can be done by reduction to normal dl-programs under the answer set semantics. In the special cases of positive and stratified loosely coupled fuzzy dl-programs, the answer set semantics coincides with a canonical least model and an iterative least model semantics, respectively, and has a characterization in terms of a fixpoint and an iterative fixpoint semantics, respectively.

Example 15. Consider the fuzzy DL knowledge base L of a car shopping Web site, which defines especially (i) the fuzzy concepts of sports cars (*SportsCar*), "at most 22 000 €" (*LeqAbout22000*), and "around 150 horse power" (*Around150HP*), (ii) the attributes of the price and of the horse power of a car (*hasInvoice* and *hasHP*, respectively), and (iii) the properties of some concrete cars (such as a *MazdaMX5Miata* and a *MitsubishiES*). Then, a loosely coupled fuzzy dl-program $KB = (L, P)$ is given by the set of fuzzy dl-rules P, which contains only the following fuzzy dl-rule encoding the request of a buyer (asking for a sports car costing at most 22 000 € and having around 150 horse power), where $\otimes$ may be the conjunction strategy of, e.g., Gödel Logic (that is, $x \otimes y = \min(x, y)$ for all $x, y \in [0, 1]$, used to evaluate the logical connectives $\wedge$ and $\leftarrow$ on truth values):

$$\begin{aligned} \mathit{query}(x) \leftarrow_{\otimes} \; & \mathit{DL}[\mathit{SportsCar}](x) \wedge_{\otimes} \mathit{DL}[\exists \mathit{hasInvoice}.\mathit{LeqAbout22000}](x) \wedge_{\otimes} \\ & \mathit{DL}[\exists \mathit{hasHP}.\mathit{Around150HP}](x) \geq 1\,. \end{aligned}$$

The above fuzzy dl-program $KB = (L, P)$ is positive, and has a minimal model M_{KB}, which defines the degree to which some concrete cars in the DL knowledge base L match the buyer's request, for example,

$$M_{KB}(\mathit{query}(\mathit{MazdaMX5Miata})) = 0.36\,, \quad M_{KB}(\mathit{query}(\mathit{MitsubishiES})) = 0.32\,.$$

That is, the *MazdaMX5Miata* is ranked top with the degree 0.36, while the *MitsubishiES* is ranked second with the degree 0.32.

Towards an infrastructure for additionally handling uncertainty in the reasoning layers of the Semantic Web, probabilistic fuzzy dl-programs [77] combine fuzzy description logics, fuzzy logic programs (with stratified default-negation), and probabilistic uncertainty in a uniform framework for the Semantic Web. Intuitively, they allow for defining several rankings on ground atoms using fuzzy vagueness, and then for merging these rankings using probabilistic uncertainty (by associating with each ranking a probabilistic weight and building the weighted sum of all rankings). Such programs also give rise to important concepts dealing with both probabilistic uncertainty and fuzzy vagueness, such as the expected truth value of a crisp sentence and the probability of a vague sentence.

Example 16. A loosely coupled probabilistic fuzzy dl-program is given by a suitable fuzzy DL knowledge base L and the following set of fuzzy dl-rules P, modeling some query reformulation / retrieval steps using ontology mapping rules:

$$\begin{aligned} \mathit{query}(x) \leftarrow_{\otimes} \; & \mathit{SportyCar}(x) \wedge_{\otimes} \mathit{hasPrice}(x, y_1) \wedge_{\otimes} \mathit{hasPower}(x, y_2) \; \wedge_{\otimes} \\ & \mathit{DL}[\mathit{LeqAbout22000}](y_1) \wedge_{\otimes} \mathit{DL}[\mathit{Around150HP}](y_2) \geq 1\,, \quad (1.40) \\ \mathit{SportyCar}(x) \leftarrow_{\otimes} \; & \mathit{DL}[\mathit{SportsCar}](x) \wedge_{\otimes} \mathit{sc}_{pos} \geq 0.9\,, \quad (1.41) \\ \mathit{hasPrice}(x, y) \leftarrow_{\otimes} \; & \mathit{DL}[\mathit{hasInvoice}](x, y) \wedge_{\otimes} \mathit{hi}_{pos} \geq 0.8\,, \quad (1.42) \\ \mathit{hasPower}(x, y) \leftarrow_{\otimes} \; & \mathit{DL}[\mathit{hasHP}](x, y) \wedge_{\otimes} \mathit{hhp}_{pos} \geq 0.8\,, \quad (1.43) \end{aligned}$$

where we assume the set $C = \{\{\mathit{sc}_{pos}, \mathit{sc}_{neg}\}, \{\mathit{hi}_{pos}, \mathit{hi}_{neg}\}, \{\mathit{hhp}_{pos}, \mathit{hhp}_{neg}\}\}$ of values of random variables and the probability distribution μ on all joint instantiations of these variables, given by $\mu\colon \mathit{sc}_{pos}, \mathit{sc}_{neg}, \mathit{hi}_{pos}, \mathit{hi}_{neg}, \mathit{hhp}_{pos}, \mathit{hhp}_{neg} \mapsto 0.91, 0.09, 0.78, 0.22, 0.83, 0.17$ under probabilistic independence. Rule

(1.40) is the buyer's request, but in a "different" terminology than the one of the car selling site. Rules (1.41)–(1.43) are so-called ontology alignment mapping rules. For example, rule (1.41) states that the predicate "SportyCar" of the buyer's terminology refers to the concept "SportsCar" of the selected site with probability 0.91.

The following may be some tight consequences of the above probabilistic fuzzy dl-program (where for ground atoms q, we use $(\mathbf{E}[q])[L, U]$ to denote that the expected truth value of q lies in the interval $[L, U]$):

$$(\mathbf{E}[\mathit{query}(\mathit{MazdaMX5Miata})])[0.21, 0.21]\,,\quad (\mathbf{E}[\mathit{query}(\mathit{MitsubishiES})])[0.19, 0.19]\,.$$

That is, the *MazdaMX5Miata* is ranked first with the degree 0.21, while the *MitsubishiES* is ranked second with the degree 0.19.

Tightly coupled fuzzy dl-programs under the answer set semantics [78, 80] are a tight integration of fuzzy disjunctive logic programs under the answer set semantics with fuzzy description logics. From a different perspective, they are a generalization of tightly coupled disjunctive dl-programs by fuzzy vagueness in both the description logic and the logic program component. This is the first approach to fuzzy dl-programs that may contain disjunctions in rule heads. Query processing in such programs can essentially be done by a reduction to tightly coupled disjunctive dl-programs. A closely related work [79] explores the problem of evaluating ranked top-k queries. It shows in particular how to compute the top-k answers in data-complexity tractable tightly coupled fuzzy dl-programs.

Example 17. A tightly coupled fuzzy dl-program $KB = (L, P)$ is given by a suitable fuzzy DL knowledge base L and the set of fuzzy rules P, which contains only the following fuzzy rule (where $x \otimes y = \min(x, y)$):

$$\begin{aligned}\mathit{query}(x) \leftarrow_{\otimes}\ & \mathit{SportyCar}(x) \wedge_{\otimes} \mathit{hasInvoice}(x, y_1) \wedge_{\otimes} \mathit{hasHorsePower}(x, y_2) \wedge_{\otimes} \\ & \mathit{LeqAbout22000}(y_1) \wedge_{\otimes} \mathit{Around150}(y_2) \geqslant 1\,.\end{aligned}$$

Informally, *query* collects all sports cars, and ranks them according to whether they cost at most around 22 000 € and have around 150 HP. Another fuzzy rule involving also a negation in its body and a disjunction in its head is given as follows (where $\ominus x = 1 - x$ and $x \oplus y = \max(x, y)$):

$$\begin{aligned}\mathit{Small}(x) \vee_{\oplus} \mathit{Old}(x) \leftarrow_{\otimes}\ & \mathit{Car}(x) \wedge_{\otimes} \mathit{hasInvoice}(x, y) \wedge_{\otimes} \\ & \mathit{not}_{\ominus}\, \mathit{GeqAbout15000}(y) \geqslant 0.7\,.\end{aligned}$$

This rule says that a car costing at most around 15 000 € is either small or old. Notice here that *Small* and *Old* may be two concepts in the fuzzy DL knowledge base L. That is, the tightly coupled approach to fuzzy dl-programs under the answer set semantics also allows for using the rules in P to express relationships between the concepts and roles in L. This is not possible in the loosely coupled approach to fuzzy dl-programs under the answer set semantics in [72, 76], since the dl-queries there can only occur in rule bodies, but not in rule heads.

1.4.3 CQ-Programs

An extension for dl-programs are cq-programs [41], which allow for expressing *(union of) conjunctive queries* (U)CQ over description logics in the dl-atoms, and disjunctions in the head of the rules.

This approach for hybrid reasoning with rules and ontologies is following the loose coupling approach, i.e., it is a heterogeneous integration that differentiates between logic programming predicates and description logic concept and roles. cq-programs benefit of some of the advantages of the loose coupling approach, such as the possibility of immediate integration of existing solvers for the implementation of the language. Also, the clear separation of the involved components enables the possibility of designing a modular architecture, as may be imagined.

In contrast with dl-programs, the cq-program combination is tighter in a sense that it allows to existentially quantify over unknown individuals that are implicit in a DL knowledge base.

Example 18. Consider the following simplified version of a scenario in [86].

$$L = \begin{Bmatrix} \mathit{hates}(\mathit{Cain}, \mathit{Abel}), \mathit{hates}(\mathit{Romulus}, \mathit{Remus}), \\ \mathit{father}(\mathit{Cain}, \mathit{Adam}), \mathit{father}(\mathit{Abel}, \mathit{Adam}), \\ \mathit{father} \sqsubseteq \mathit{parent}, \\ \exists \mathit{father}.\exists \mathit{father}^{-}.\{\mathit{Remus}\}(\mathit{Romulus}) \end{Bmatrix}$$

$$P = \{\mathit{BadChild}(X) \leftarrow \mathrm{DL}[\mathit{parent}](X, Z), \mathrm{DL}[\mathit{parent}](Y, Z), \mathrm{DL}[\mathit{hates}](X, Y)\}$$

Apart from the explicit facts, L states that each *father* is also a *parent* and that *Romulus* and *Remus* have a common father. The single rule in P specifies that an individual hating a sibling is a *BadChild*. From this dl-program, $\mathit{BadChild}(\mathit{Cain})$ can be concluded, but not $\mathit{BadChild}(\mathit{Romulus})$.

Instead of P, let us use

$$P' = \{\mathit{BadChild}(X) \leftarrow \mathrm{DL}[\mathit{parent}(X, Z), \mathit{parent}(Y, Z), \mathit{hates}(X, Y)](X, Y)\},$$

where the body of the rule is a CQ $\{\mathit{parent}(X, Z), \mathit{parent}(Y, Z), \mathit{hates}(X, Y)\}$ to L with distinguished variables X and Y. We then obtain the desired result; that is, we can derive the fact $\mathit{BadChild}(\mathit{Romulus})$.

The semantics of the cq-programs is in spirit of dl-programs, and mainly differs in the generalized entailment notion for cq-atoms, which extend that of dl-atoms. Informally, a cq-atom α is in form $\mathrm{DL}[\lambda; q](\mathbf{X})$, where q can be a union of conjunctive queries with output variables $\mathbf{X}$, while λ represents a list of modifiers for the description logic base L at hand, with the same meaning given in dl-programs. The CQ-extension adds additional expressivity to dl-programs, as is evident by results that show an increase in complexity from NEXP to 2-EXP for the description logic $\mathcal{SHIF}(\mathbf{D})$.

A further plus of this extension is that it opens the floodgates for exploiting optimizations in dl-programs, via a technique able to produce rewritten programs where the computational burden can be shifted to and from one of the two reasoners at hand. For instance, conjunctions of atoms can be computed, whenever

semantically equivalent, on the description logic base side instead that on the logic program side. In [41], several forms of optimizing rewriting rules have been defined to rewrite DL-queries in rule bodies to more efficient ones. Experimental results comparing unoptimized to rewritten programs show a substantial performance improvement.

1.5 Conclusion

In this chapter, we have briefly shown work that has been done in REWERSE on the issue of combining rules and ontologies. To this end, we have first given an overview of different combination approaches, which have been systematically grouped into a classification that takes different degree of integration and of rules and ontologies into account.

We have then presented the two streams of genuine approaches which have been pursed in REWERSE by the groups in Vienna and Linköping, respectively, to give meaningful and expressive combinations that faithfully generalize the stable models and the well-founded semantics of logic programs, respectively, leading to nonmonotonic combinations of rules and ontologies whose prototype implementations reflected the state of the art in this area. Furthermore, several extensions to these approaches have been briefly discussed, which address needs such as handling probabilistic information, fuzzy values, or more expressive queries to ontologies than simple instance checks or consistency tests.

While the work on combinations of rules and ontologies in REWERSE has broken new ground and was fruitfully taken up by other groups within REWERSE but also outside (in particular, HEX-programs and dlvhex have found applications in various contexts), its impact on the development of the rules layer of the Semantic Web, and in particular to emerging standards, has yet to materialize. The reason is that different from ontologies, the standardization of rules that is targeted by the RIF working group of the W3C (see Section 1.2.4) is a formidable challenge, given that there are very many notions of rules and their semantics; this is one of the reasons that at the time of this writing, merely a compromise for a core rule dialect (RIF-BLD) is what has been achieved so far; features such as negation (even stratified one) have been targeted in more comprehensive packages, but not realized so far. We expect that stable models and the well-founded semantics will be the premier semantics that are reflected in a RIF standard for non-monotonic negation that is beyond stratified negation in logic programs, and that the ideas and concepts which have been developed in the REWERSE streams will impact on the definition of possible interfacing between rules and ontologies in the emerging standards.

At present, the issue of combining rules and ontologies for the Semantic Web is not regarded to be satisfactorily solved; a number of different approaches have been made so far, but they all have some features that do not suggest them to be regarded the ultimate solution to the problem; let alone that perhaps there is no single, "universal" such solution, but a range of different solutions which cater different features and needs that have to be fulfilled in different contexts.

This already manifests in different types of rules; in REWERSE, the focus was on logic-based rules, but other rule types such as production rules are equally important and require different treatment; in fact, an integration of production rules and ontologies with bidirectional information flow is an interesting subject for future work, in which the operational and logical semantics of the rules and the ontology, respectively, have to be bridged. The OntoRule project will within the "7th Framework Program" (FP7) of the EU Commission target business rules and policies, and will to a great deal be based on a lower layer that integrates production rules and ontologies, aside with logic programming rules. By way of this project, results of REWERSE will migrate more towards practical exploitation and into commercial rules engines.

In order to make expressive combinations of rules and ontologies available for deployment to applications, a number of research tasks remain to be pursued.

Currently, we lack extended case studies and large scale examples beyond the toy examples that have been considered in the seminal papers that introduced the approaches. Such case studies might provide helpful insight and give some guidance in the development of a "gold standard" for rules plus ontologies. At the least, required constructs in the language, be they just syntactic sugar or really increasing the expressiveness of a formalism, should be identifiable in this way. The trouble is, however, to single out a set of representative cases, which is by no means trivial. A benchmark suite would be very valuable and, if carefully composed, undoubtedly an important step forward.

Another issue are complex data structures, and realizations of the combinations beyond the Datalog fragment. Indeed, in practice, one needs to handle complex data that are aggregations of other data, such as records, lists, sets, etc. Such data structures can be modeled in many logic programming systems using function symbols, and support in terms of explicit syntax is offered. However, in the current solvers for stable model and answer set semantics, function symbols are largely banned because they are a well-known source of undecidability, even in rather plain settings; only more recently, work on decidable classes and prototype implementations of stable models semantics with function symbols has been carried out (cf. [15, 14, 24, 100] and references therein), and function symbols also increasingly attract attention as a modeling construct. The DLV-Complex system [23, 24] aims at providing functions symbols in a decidable setting, giving support to lists, sets along with libraries for their manipulations. It remains to see how logic programs in this setting can be combined with ontologies; semantically, the gap between rules and ontologies widens by the use of such function symbols, and decidability issues has to be reconsidered.

An obvious task is the development of better algorithms and efficient implementations. The current prototype implementations serve more as proofs of concept and experimental testbeds, but are not largely optimized. There is a lot of room for improvement, even though the optimization methods are expected to be tailored to a particular semantics and implementation setting. The intertwining of a rules and an ontology engine, as done in the prototype implementations of dl-programs and HD-rules, imposes specific requirements that can not be

easily transferred to other implementations. Developing an integrated engine that processes rules and ontologies en par is an interesting issue; whether a conversion of logic programs into ontology axioms or vice versa a mapping of ontologies into logic programming rules is a viable approach remains to be explored. This, however, may work well for fragments of combinations in which such conversions are easily possible.

In close connection to the previous issue are semantic and computational properties of combinations. There is clearly a trade-off between the expressiveness of a formalism on the one hand and its intrinsic complexity on the other. If we expect to have fast reasoning over knowledge bases with large extensional part, comprising millions (or even billions) of facts, then naturally the reasoning tasks per se must not have high intrinsic complexity. For this reason, it is important to have an understanding of the complexity characteristics of combinations, to know about fragments with tractable and low complexity (just polynomial time as such might not be sufficient for practical applications, if the data volume is large), and to respect such characteristics in implementations in a way that easy instances are solved with little effort while more computation time is spent on harder instances. Recent research on rules and conjunctive query answering over description logics from the lower expressiveness end like $\mathcal{EL}$ and $\mathcal{EL}$++ [6, 7], or DL-Lite [25, 92] may be here a starting point.

Finally, an important issue is also to combine knowledge sources beyond rules and ontologies. Indeed, a rule base and an ontology may be just two components in an information system that consists of many other components that are in different formats. And while throughout this chapter, the rules and the ontology have been considered as more or less integral parts of one description, this picture may no longer be valid if the components are independently conceived and autonomous, like they happen to be in a peer to peer system. In such a case, also the viewpoint of semantic combination should be rather different, and incorporating trust is an important requirement.

Acknowledgments

The work in this chapter was partially supported by the European Commission through the project REWERSE (IST-2003-506779), in which the main technical results were obtained, and the project Ontorule (FP7 231875), as well as by the Austrian Science Fund (FWF) grants P17212, P20840, and P20841. Thomas Lukasiewicz has been supported by the German Research Foundation (DFG) under the Heisenberg Programme.

References

1. Analyti, A., Antoniou, G., Damásio, C.V., Wagner, G.: Stable model theory for extended RDF ontologies. In Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A., eds.: The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings. Volume 3729 of Lecture Notes in Computer Science., Springer (2005) 21–36

2. Antoniou, G., Maher, M., Billington, D.: Defeasible Logic versus Logic Programming. Journal of Logic Programming **41**(1) (2000) 45–57
3. Antoniou, G., Bikakis, A.: DR-Prolog: A system for defeasible reasoning with rules and ontologies on the Semantic Web. IEEE Trans. Knowl. Data Eng. **19**(2) (2007) 233–245
4. Antoniou, G., Damásio, C.V., Grosof, B., Horrocks, I., Kifer, M., Maluszynski, J., Patel-Schneider, P.F.: Combining rules and ontologies: A survey. Tech. Rep. IST506779/Linköping/I3-D3/D/PU/a1, Linköping University (Feb. 2005)
5. Apt, K.R., Bol, R.N.: Logic programming and negation: A survey. J. Log. Program. **19/20** (1994) 9–71
6. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05, Edinburgh, UK, Morgan-Kaufmann Publishers (2005)
7. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope further. In Clark, K., Patel-Schneider, P.F., eds.: In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions. (2008)
8. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. 2nd Edition. Cambridge University Press (2007)
9. Baader, F., Hollunder, B.: Embedding Defaults into Terminological Knowledge Representation Formalisms. Journal of Automated Reasoning **14**(1) (February 1995) 149–180
10. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
11. Baral, C., Gelfond, M.: Logic Programming and Knowledge Representation. Journal of Logic Programming **19/20** (1994) 73–148
12. Boley, H., (eds.), M.K.: RIF Basic Logic Dialect (July 2008) W3C Working Draft, available at `http://www.w3.org/TR/2008/WD-rif-bld-20080730/`.
13. Boley, H., Kifer, M., Pătrânjan, P.L., Polleres, A.: Rule interchange on the web. In: Reasoning Web 2007. Volume 4636 of Lecture Notes in Computer Science. Springer (SEP 2007) 269–309
14. Bonatti, P., Baselice, S.: Composing normal programs with function symbols. In de La Banda, M., Pontelli, E., eds.: Proceedings 24th International Conference on Logic Programming (ICLP 2008). Number 5366 in LNCS, Springer (2008) 425–439
15. Bonatti, P.A.: Reasoning with infinite stable models. Artificial Intelligence **156**(1) (2004) 75–111
16. Bonatti, P.A., Eiter, T., Faella, M.: Advanced Policy Queries. REWERSE Deliverable I2-D6, Dipartimento di Scienze Fisiche - Sezione di Informatica, University of Naples "Federico II" (2006)
17. Buccafurri, F., Leone, N., Rullo, P.: Strong and Weak Constraints in Disjunctive Datalog. In Dix, J., Furbach, U., Nerode, A., eds.: Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97). Volume 1265 of Lecture Notes in AI (LNAI)., Dagstuhl, Germany, Springer Verlag (July 1997) 2–17
18. Calì, A., Lukasiewicz, T.: Tightly integrated probabilistic description logic programs for the Semantic Web. In: Proceedings ICLP-2007. Volume 4670 of LNCS., Springer (2007) 428–429
19. Calì, A., Lukasiewicz, T.: An approach to probabilistic data integration for the Semantic Web. In: Uncertainty Reasoning for the Semantic Web I. Volume 5327 of LNCS., Springer (2008) 52–65

20. Calì, A., Lukasiewicz, T., Predoiu, L., Stuckenschmidt, H.: Rule-based approaches for representing probabilistic ontology mappings. In: Uncertainty Reasoning for the Semantic Web I. Volume 5327 of LNCS., Springer (2008) 66–87
21. Calì, A., Lukasiewicz, T., Predoiu, L., Stuckenschmidt, H.: Tightly integrated probabilistic description logic programs for representing ontology mappings. In: Proceedings FoIKS-2008. Volume 4932 of LNCS., Springer (2008) 178–198
22. Calì, A., Lukasiewicz, T., Predoiu, L., Stuckenschmidt, H.: Tightly coupled probabilistic description logic programs for the Semantic Web. Journal on Data Semantics XII (2009)
23. Calimeri, F., Cozza, S., Ianni, G.: External sources of knowledge and value invention in logic programming. Annals of Mathematics and Artificial Intelligence **50**(3-4) (2007) 333–361
24. Calimeri, F., Cozza, S., Ianni, G., Leone, N.: Computable functions in ASP: Theory and implementation. In de La Banda, M., Pontelli, E., eds.: Proceedings 24th International Conference on Logic Programming (ICLP 2008). Number 5366 in LNCS, Springer (2008) 407–424
25. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Journal of Automated Reasoning **39**(3) (2007) 385–429
26. Chen, W., Kifer, M., Warren, D.S.: Hilog: A foundation for higher-order logic programming. Journal of Logic Programming **15**(3) (February 1993) 187–230
27. Clark, K.L.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Databases. Plenum Press (1978) 293–322
28. Dao-Tran, M., Eiter, T., Krennwallner, T.: Realizing Default Logic over Description Logic Knowledge Bases. In: Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2009), Springer (2009) To appear.
29. de Bruijn, J., Eiter, T., Polleres, A., Tompits, H.: Embedding non-ground logic programs into autoepistemic logic for knowledge-base combination. In Veloso, M.M., ed.: IJCAI. (2007) 304–309
30. de Bruijn, J., Eiter, T., Tompits, H.: Embedding approaches to combining rules and ontologies into autoepistemic logic. In Brewka, G., Lang, J., eds.: Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2008), AAAI Press (2008) 485–495
31. de Bruijn, J., Pearce, D., Polleres, A., Valverde, A.: Quantified Equilibrium Logic and Hybrid Rules. In: Proc. of International Conference on Web Reasoning and Rule Systems (RR'07). Volume 4524 of Lecture Notes in Computer Science., Springer (2007) 58–72
32. Dean, M., Schreiber, G., Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference (February 2004) W3C Recommendation.
33. Donini, F., Lenzerini, M., Nardi, D., Schaerf, A.: AL-Log: Integrating Datalog and description logics. Intelligent Information Systems **10(3)** (1998) 227–252
34. Drabent, W.: SLS-resolution without floundering. In Pereira, L.M., Nerode, A., eds.: Proc. 2nd International Workshop on Logic Programming and Non-Monotonic Reasoning, MIT Press (June 1993) 82–98
35. Drabent, W.: What is failure? An approach to constructive negation. Acta Informatica **32**(1) (February 1995) 27–59
36. Drabent, W., Maluszynski, J.: Hybrid Rules with Well-Founded Semantics. Preliminarily accepted to Knowledge and Information Systems (2009)

37. Drabent, W., Henriksson, J., Maluszynski, J.: HD-rules: A hybrid system interfacing Prolog with DL-reasoners. In: Proceedings of the ICLP'07 Workshop on Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services (ALPSWS2007). CEUR Workshop Proceedings Vol 287 (2007) http://www.ceur-ws.org/Vol-287.
38. Drabent, W., Henriksson, J., Maluszynski, J.: Hybrid reasoning with rules and constraints under well-founded semantics. In Marchiori, M., Pan, J.Z., de Sainte Marie, C., eds.: RR. Volume 4524 of Lecture Notes in Computer Science., Springer (2007) 348–357
39. Drabent, W., Małuszyński, J.: Well-founded Semantics for Hybrid Rules. In: Proc. of International Conference on Web Reasoning and Rule Systems (RR'07). Volume 4524 of Lecture Notes in Computer Science., Springer (2007) 1–15
40. Eiter, T., Ianni, G., Krennwallner, T., Polleres, A.: Rules and ontologies for the Semantic Web. In Baroglio, C., Bonatti, P.A., Maluszynski, J., Marchiori, M., Polleres, A., Schaffert, S., eds.: Reasoning Web: 4th International Summer School 2008, Venice Italy, September 7-11, 2008, Tutorial Lectures. Volume 5224 of LNCS. Springer (September 2008) 1–53 Slides available at http://rease.semanticweb.org/.
41. Eiter, T., Ianni, G., Krennwallner, T., Schindlauer, R.: Exploiting conjunctive queries in description logic programs. Annals of Mathematics and Artificial Intelligence (2009) Published online 27 January 2009 (doi:10.1007/s10472-009-9111-3). Also available as Tech. Rep. INFSYS RR-1843-08-02, Inst. of Information Systems, TU Vienna.
42. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R.: Well-founded semantics for description logic programs in the Semantic Web. Technical Report INFSYS RR-1843-09-01, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria (March 2009)
43. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programminag with description logics for the Semantic Web. Artificial Intelligence **172**(12-13) (2008) 1495–1539
44. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In: International Joint Conference on Artificial Intelligence (IJCAI) 2005, Edinburgh, UK (August 2005) 90–96
45. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective integration of declarative rules with external evaluations for semantic-web reasoning. In Sure, Y., Domingue, J., eds.: ESWC. Volume 4011 of Lecture Notes in Computer Science., Springer (2006) 273–287
46. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web. In Dubois, D., Welty, C., Williams, M.A., eds.: Proceedings Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), June 2-5, Whistler, British Columbia, Canada, Morgan Kaufmann (2004) 141–151
47. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Well-founded semantics for description logic programs in the Semantic Web. In: Proceedings RuleML-2004. Volume 3323 of LNCS., Springer (2004) 81–97
48. Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T., Ielpa, G.: Design and Implementation of Aggregate Functions in the DLV System. Theory and Practice of Logic Programming **8**(5–6) (November 2008) 545–580
49. Ferrand, G., Deransart, P.: Proof method of partial correctness and weak completeness for normal logic programs. J. Log. Program. **17**(2/3&4) (1993) 265–278

50. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R.A., Bowen, K., eds.: Proceedings of the Fifth International Conference on Logic Programming, Cambridge, Massachusetts, The MIT Press (1988) 1070–1080
51. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Logic Programming: Proceedings Fifth Intl Conference and Symposium, Cambridge, Mass., MIT Press (1988) 1070–1080
52. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9** (1991) 365–385
53. Grosof, B., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logic. In: Proceedings of 12th International Conference on the World Wide Web. (2003)
54. Grosof, B.N.: Prioritized Conflict Handling for Logic Programs. In Małuszyński, J., ed.: Logic Programming, Proceedings of the 1997 International Symposium, MIT Press (1997) 197–211
55. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition **5**(2) (1993) 199–220
56. Herre, H., Jaspars, J., Wagner, G.: Partial logics with two kinds of negation as a foundation for knowledge-based reasoning. In Gabbay, D., Wansing, H., eds.: What is Negation? Kluwer Academic Publishers (1999) 121–159
57. Heymans, S., de Bruijn, J., Predoiu, L., Feier, C., Nieuwenborgh, D.V.: Guarded hybrid knowledge bases. TPLP **8**(3) (2008) 411–429
58. Heymans, S., Toma, I.: Ranking services using fuzzy hex-programs. In Calvanese, D., Lausen, G., eds.: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems (RR 2008), Karlsruhe, Germany, October 31-November 1, 2008. Volume 5341 of LNCS., Springer (2008) 181–196
59. Hoehndorf, R., Loebe, F., Kelso, J., Herre, H.: Representing default knowledge in biomedical ontologies: Application to the integration of anatomy and phenotype ontologies. BMC Bioinformatics **8**(1) (2007) 377
60. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proceedings of the 10th International Conference of Knowledge Representation and Reasoning (KR-2006). (2006) 57–67
61. Kifer, M.: Rules and Ontologies in F-Logic. In Eisinger, N., Małuszyński, J., eds.: Reasoning Web. Volume 3564 of Lecture Notes in Computer Science., Springer (2005) 22–34
62. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. J. ACM **42**(4) (1995) 741–843
63. Knorr, M., Alferes, J.J., Hitzler, P.: A Well-founded Semantics for Hybrid MKNF Knowledge Bases. In: DL'07, 20th Int. Workshop on Description Logics, Bozen-Bolzano University Press (2007) 347–354
64. Knorr, M., Alferes, J.J., Hitzler, P.: A coherent well-founded model for hybrid mknf knowledge bases. In Ghallab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N.M., eds.: ECAI. Volume 178 of Frontiers in Artificial Intelligence and Applications., IOS Press (2008) 99–103
65. Konolige, K.: Quantification in autoepistemic logic. Fundam. Inform. **15**(3-4) (1991) 275–300
66. Krötzsch, M., Rudolph, S., Hitzler, P.: Description logic rules. In Ghallab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N.M., eds.: ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings. Volume 178 of Frontiers in Artificial Intelligence and Applications., IOS Press (2008) 80–84

67. Krötzsch, M., Rudolph, S., Hitzler, P.: Elp: Tractable rules for OWL 2. In Sheth, A., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K., eds.: Proceedings of the 7th International Semantic Web Conference (ISWC 2008). Volume 5318 of LNCS., Springer (OCT 2008) 649–664
68. Levy, A., Rousset, M.: CARIN: A representation language combining Horn rules and description logics. Artificial Intelligence 104(1 2):165 209 (1998)
69. Lifschitz, V.: Nonmonotonic databases and epistemic queries. In: Proceedings IJCAI-91. (1991) 381–386
70. Lloyd, J.W.: Foundations of logic programming. Second extended edn. Springer series in symbolic computation. Springer-Verlag New York, Inc. (1987)
71. Lukasiewicz, T.: Probabilistic description logic programs. In: Proceedings ECSQARU-2005. Volume 3571 of LNCS., Springer (2005) 737–749
72. Lukasiewicz, T.: Fuzzy description logic programs under the answer set semantics for the Semantic Web. In: Proceedings RuleML-2006, IEEE Computer Society (2006) 89–96
73. Lukasiewicz, T.: A novel combination of answer set programming with description logics for the Semantic Web. In: Proceedings ESWC-2007. Volume 4519 of LNCS., Springer (2007) 384–398
74. Lukasiewicz, T.: Probabilistic description logic programs. Int. J. Approx. Reasoning **45**(2) (2007) 288–307
75. Lukasiewicz, T.: Tractable probabilistic description logic programs. In: Proceedings SUM-2007. Volume 4772 of LNCS., Springer (2007) 143–156
76. Lukasiewicz, T.: Fuzzy description logic programs under the answer set semantics for the Semantic Web. Fundam. Inform. **82**(3) (2008) 289–310
77. Lukasiewicz, T., Straccia, U.: Description logic programs under probabilistic uncertainty and fuzzy vagueness. In: Proceedings ECSQARU-2007. Volume 4724 of LNCS., Springer (2007) 187–198
78. Lukasiewicz, T., Straccia, U.: Tightly integrated fuzzy description logic programs under the answer set semantics for the Semantic Web. In: Proceedings RR-2007. Volume 4524 of LNCS., Springer (2007) 289–298
79. Lukasiewicz, T., Straccia, U.: Top-k retrieval in description logic programs under vagueness for the Semantic Web. In: Proceedings SUM-2007. Volume 4772 of LNCS., Springer (2007) 16–30
80. Lukasiewicz, T., Straccia, U.: Tightly coupled fuzzy description logic programs under the answer set semantics for the Semantic Web. Int. J. Semantic Web Inf. Syst. **4**(3) (2008) 68–89
81. Małuszyński, J.: Integration of Rules Ontologies. In Liu, L., Özsu, M.T., eds.: Encyclopedia of Database Systems. Springer (2009) Entry.
82. Marin, D.: A formalization of RDF. Technical Report TR/DCC-2006-8, TR Dept. Computer Science, Universidad de Chile (2006)
83. Marriott, K., Stuckey, P.J., Wallace, M.: Constraint logic programming. In: Handbook of Constraint Programming. Elsevier (2006)
84. Miller, L., Brickley, D.: The Friend of a Friend (FOAF) Project (since 2000) `http://www.foaf-project.org/`.
85. Motik, B., Rosati, R.: A faithful integration of description logics with logic programming. In: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence. (2007) 477–482
86. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. J. Web Sem. **3**(1) (2005) 41–60

87. Motik, B., Rosati, R.: Closing semantic web ontologies. Technical report, University of Manchester (2006) Version March 7, 2007. Available at `http://web.comlab.ox.ac.uk/people/Boris.Motik/pubs/mr06closing-report.pdf`.
88. Nieuwenborgh, D.V., Cock, M.D., Vermeir, D.: Computing Fuzzy Answer Sets Using dlvhex. In Dahl, V., Niemelä, I., eds.: Proceedings of the 23rd International Conference on Logic Programming (ICLP 2007), Porto, Portugal, September 8-13, 2007. Volume 4670 of LNCS., Springer (2007) 449–450
89. Nieuwenborgh, D.V., Eiter, T., Vermeir, D.: Conditional Planning with External Functions. In Baral, C., Brewka, G., Schlipf, J.S., eds.: Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007). Volume 4483 of LNCS., Springer (2007) 214–227
90. Nilsson, U., Małuszyński, J.: Logic, Programming and Prolog. Second edn. John Wiley and Sons (1995) now available free of charge at `http://www.ida.liu.se/~ulfni/lpp/`.
91. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax (February 2004) W3C Recommendation.
92. Poggi, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Linking data to ontologies. Journal of Data Semantics **X** (2008) 133–173
93. Przymusinski, T.C.: On the declarative and procedural semantics of logic programs. Journal of Automated Reasoning **5** (1989) 167–205
94. Reiter, R.: On Closed World Data Bases. In Gallaire, H., Minker, J., eds.: Logic and Data Bases. Plenum Press, New York (1978) 55–76
95. Reiter, R.: A Logic for Default Reasoning. Artificial Intelligence **13**(1–2) (1980) 81–132
96. Rosati, R.: On the decidability and complexity of integrating ontologies and rules. Journal of Web Semantics **3**(1) (2005) 61–73
97. Rosati, R.: $\mathcal{DL}$+*log*: Tight integration of Description Logics and disjunctive Datalog. In Doherty, P., Mylopoulos, J., Welty, C.A., eds.: KR, AAAI Press (2006) 68–78
98. Ross, K.A.: Modular stratification and magic sets for datalog programs with negation. J. ACM **41**(6) (1994) 1216–1266
99. Schindlauer, R.: Answer-Set Programming for the Semantic Web. PhD thesis, Vienna University of Technology, Austria (December 2006)
100. Simkus, M., Eiter, T.: FDNC: Decidable non-monotonic disjunctive logic programs with function symbols. In: Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR2007). Volume 4790 of Lecture Notes in Computer Science., Springer (2007) 514–530 Full paper to appear in *ACM TOCL*.
101. Sintek, M., Decker, S.: TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In: International Semantic Web Conference (ISWC), Sardinia (June 2002) Available at `http://triple.semanticweb.org/`.
102. van Gelder, A., Ross, K.A., Schlipf, J.S.: Unfounded Sets and Well-founded Semantics for General Logic Programs. In: Principles of Database Systems, ACM (1988) 221–230
103. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The Well-Founded Semantics for General Logic Programs. Journal of the ACM **38**(3) (1991) 620–650