Semantic Web Reasoning with Conceptual Logic Programs

Stijn Heymans, Davy Van Nieuwenborgh*, and Dirk Vermeir**

Dept. of Computer Science
Vrije Universiteit Brussel, VUB
Pleinlaan 2, B1050 Brussels, Belgium
{sheymans,dvnieuwe,dvermeir}@vub.ac.be

Abstract. We extend Answer Set Programming with, possibly infinite, open domains. Since this leads, in general, to undecidable reasoning, we restrict the syntax of programs, while carefully guarding useful knowledge representation mechanisms such as negation as failure and inequalities. Reasoning with the resulting Conceptual Logic Programs can be reduced to finite, normal Answer Set Programming, for which reasoners are available.
We argue that Conceptual Logic Programming is a useful tool for uniformly representing and reasoning with both ontologies and rules on the Semantic Web, as they can capture a large fragment of the OWL DL ontology language, while extending it in various aspects.

1 Introduction

Ontology languages such as OWL and OWL DL[5] are set to play a vital role on the future Semantic Web, as they are designed to represent a wide range of knowledge on the Web and to ensure decidable reasoning with it. Decidability of such languages often results from the decidability of the underlying Description Logic (DL)[4] that defines its formal semantics, e.g., the DL $\mathcal{SHOIQ}(\mathbf{D})$ is the DL corresponding to OWL DL.

Another well-established knowledge representation formalism is Answer Set Programming (ASP)[11], a Logic Programming (LP) paradigm that captures knowledge by programs whose answer sets express the intended meaning of this knowledge. The answer set semantics presumes that all relevant domain elements are present in the program. Such a closed domain assumption is, however, problematic if one wishes to use ASP for ontological reasoning since ontologies describe knowledge in terms of concepts and interrelationships between them, and are thus mostly independent of constants.

E.g., consider the knowledge that managers drive big cars, that one is either a manager or not, and that Felix is definitely not a manager. This is represented by the program

* Supported by the FWO.
** This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-37004 WASP project.

G. Antoniou and H. Boley (Eds.): RuleML 2004, LNCS 3323, pp. 113-127, 2004.

P_1:

$$bigCar(X) \leftarrow Manager(X)$$
$$Manager(X) \vee not\ Manager(X) \leftarrow$$
$$\neg Manager(felix) \leftarrow$$

Using traditional ASP, grounding would yield the program

$$bigCar(felix) \leftarrow Manager(felix)$$
$$Manager(felix) \vee not\ Manager(felix) \leftarrow$$
$$\neg Manager(felix) \leftarrow$$

which has a single answer set $\{\neg Manager(felix)\}$ such that one would wrongfully conclude that there are never managers or persons that drive big cars.

We resolve this by introducing, possibly infinite, *open domains*. Under the *open answer set semantics* the example has an answer set $(\mathcal{H} = \{felix, heather\}, M = \{\neg Manager(felix), Manager(heather), bigCar(heather)\})$ where $\mathcal{H}$ is a *universe* for P_1 that extends the constants present in P_1 and M is an answer set of P_1 grounded with $\mathcal{H}$. One would rightfully conclude that it is possible that there are persons that are managers and thus drive big cars. Note the use of disjunction and negation as failure in the head of $Manager(X) \vee not\ Manager(X) \leftarrow$. Such rules will be referred to as *free rules* since they allow for the free introduction of literals; answer sets are, consequently, not subset minimal.

The catch is that reasoning, i.e. satisfiability checking of a predicate, with *open domains* is, in general, undecidable. In order to regain decidability, we restrict the syntax of programs while retaining useful knowledge representation tools such as negation as failure and inequality. Moreover, the result, *(local) Conceptual Logic Programs (CLPs)*, ensures a reduction of reasoning to finite, closed, ASP, making CLPs amenable for reasoning with existing answer set solvers.

As opposed to the CLPs in [16, 15], we support constants in this paper. Constants in a CLP have the effect that the tree-model property, a decidability indicator, is replaced by the more general forest-model property. Furthermore, [16, 15] characterized reasoning with CLPs by checking non-emptiness of two-way alternating tree-automata[31]. Although such automata are elegant theoretical tools, they are of little practical use, hence the importance of an identification of CLPs that can be reduced to traditional ASP.

Conceptual logic programs prove to be suitable for Semantic Web reasoning, for we can simulate an expressive DL closely related to the ontology language OWL DL. Since CLPs, as a LP paradigm, are also a natural framework for representing rule-based knowledge, they present a unifying framework for reasoning with ontologies and rules. Some additional benefits of CLPs, compared with OWL DL, are their ability to close the domain at will and to succinctly represent knowledge that is not trivially expressible using OWL DL. Finally, several query problems, in the context of databases satisfying ontologies, can be stated as satisfiability problems w.r.t. CLPs and are consequently decidable.

The remainder of the paper is organized as follows. In Section 2, we extend ASP with open domains, and in Section 3, we define (local) CLPs and reduce reasoning to normal ASP. In Section 4, we show the simulation of an expressive class of DLs and

discuss benefits of using CLPs for Semantic Web reasoning. Section 5 relates other work to our approach. Finally, Section 6 contains conclusions and directions for further research. Due to space restrictions, proofs have been omitted; they can be found in [14].

2 Answer Set Programming with Open Domains

Terms are *constants* or *variables*, denoted as lowercase or uppercase characters respectively. An *atom* is either a unary $q(s)$ or a binary $f(s,t)$ for predicates q and f, and terms s and t. A *literal* is an atom or an atom preceded by the classical negation symbol $\neg$. We assume $\neg\neg a = a$ for an atom a; for a set of literals α, $\neg\alpha = \{\neg l | l \in \alpha\}$, and α is *consistent* if $\alpha \cap \neg\alpha = \emptyset$. An *extended literal* is a literal l or a literal preceded by the *negation as failure* (naf) symbol *not*. A set of unary literals ranging over a common term s may be denoted as $\alpha(s)$, e.g., $\{a(s), not\ b(s)\} = \{a, not\ b\}(s)$. Similarly, a set of binary literals over (s,t) can be denoted as $\alpha(s,t)$. The *positive part* of a set of extended literals α is $\alpha^+ = \{l | l \in \alpha, l \text{ literal}\}$, while the *negative part* of α is $\alpha^- = \{l | not\ l \in \alpha\}$, e.g., $\{a, not\ b\}^+ = \{a\}$ and $\{a, not\ b\}^- = \{b\}$.

A *disjunctive logic program* (DLP) is a set of rules $\alpha \leftarrow \beta$ where α, the *head*, and β, the *body*, are sets of extended literals and $|\alpha^+| \leq 1$, i.e. the head contains at most one ordinary literal[1]. Atoms, (extended) literals, rules, and programs are *ground* if they do not contain variables. The constants appearing in a DLP P are denoted by $\mathcal{H}_P$, the unary predicates (possibly negated)[2] in P are $upreds(P) = \{l | l(x) \text{ in } P\}$, $bpreds(P)$ are the binary predicates, and $preds(P) = upreds(P) \cup bpreds(P)$. A *universe* $\mathcal{H}$ for a DLP P is any non-empty extension of $\mathcal{H}_P$, i.e. $\mathcal{H}_P \subseteq \mathcal{H}$. The *grounded* version $P_\mathcal{H}$ of a DLP P w.r.t. a universe $\mathcal{H}$ for P is the program P with all variables replaced by all possible elements from $\mathcal{H}$. $P_\mathcal{H}$ may be infinite if $\mathcal{H}$ is; we assume, however, that a grounded version $P_\mathcal{H}$ originates from a finite P.

E.g., the program P_2: $sel(I,S) \vee not\ sel(I,S) \leftarrow$; $av(i) \leftarrow$; $av(I) \leftarrow sel(I,S)$; expresses that an item is sold by a seller or not, an item is available if it has a seller, and we have a particular available item i. The constants in P_2 are $\mathcal{H}_P = \{i\}$; some of the universes for P_2 are $\mathcal{H}_1 = \{i, s\}$ or an infinite $\mathcal{H}_2 = \{i, x_1, x_2, \ldots\}$.

For a grounded P, let $\mathcal{L}_P$ be the set of literals that can be formed from P. A consistent subset of $\mathcal{L}_P$ is an *interpretation* of P. An interpretation I *satisfies* a literal l, denoted $I \models l$, if $l \in I$; an extended literal $not\ l$ is satisfied by I if $l \notin I$, and I satisfies a set α of extended literals, denoted $I \models \alpha$ iff I satisfies every element of α. A rule $r : \alpha \leftarrow \beta$, $\alpha \neq \emptyset$, in a grounded P is satisfied by I, denoted $I \models r$, if $I \models l$ for some $l \in \alpha$ whenever $I \models \beta$. If $\alpha = \emptyset$, i.e. the rule is a *constraint*, $I \models r$ iff $I \not\models \beta$. An interpretation I is a model of a grounded P if I satisfies every rule in P. For a *simple* grounded program P, i.e. not containing naf, an *answer set* of P is a subset minimal model of P. If P is not simple, we first reduce it for a particular interpretation I of P, with the *Gelfond-Lifschitz transformation*[22], to the simple *GL-reduct* $P^I = \{\alpha^+ \leftarrow \beta^+ \mid \alpha \leftarrow \beta \in P, \beta^- \cap I = \emptyset, \alpha^- \subseteq I\}$. An interpretation M of a grounded P is an *answer set* of P if M is an answer set of P^M.

[1] This restriction, which makes the GL-reduct disjunction-free, is not imposed by classical DLPs.

[2] In the future, we silently assume the "(possibly negated)" phrase.

For a DLP P, not grounded, an *open interpretation* is a pair $(\mathcal{H}, I)$ where $\mathcal{H}$ is a universe for P and I is an interpretation of $P_{\mathcal{H}}$. An *open answer set* of P is an open interpretation $(\mathcal{H}, M)$ such that M is an answer set of $P_{\mathcal{H}}$. In the following, we usually omit the "open" qualifier. A $p \in upreds(P)$ is *satisfiable* w.r.t. P iff there exists an answer set $(\mathcal{H}, M)$ of P and some $x \in \mathcal{H}$ such that $p(x) \in M$, in which case we also say that $(\mathcal{H}, M)$ satisfies p. A program P is *consistent* if it has an answer set. The associated reasoning tasks are *satisfiability checking* and *consistency checking*, where the latter can be reduced to the former by introducing a new predicate p, e.g., with a rule $p(X) \vee not\ p(X) \leftarrow$.

With a universe $\mathcal{H} = \{i, s, x\}$ for P_2 both $(\mathcal{H}, M_1 = \{av(i), sel(x, s), av(x)\})$ and $(\mathcal{H}, M_2 = \{av(i)\})$ are answer sets of P_2. Since M_1 contains $sel(x, s)$, the GL-reduct $P_{2\mathcal{H}}^{M_1}$ will contain $sel(x, s) \leftarrow$, which in turn motivates the presence of $sel(x, s)$ in M_1. On the other hand, since $sel(x, s) \notin M_2$, the rule $sel(x, s) \vee not\ sell(x, s) \leftarrow$ is automatically satisfied and will not be considered for inclusion in the GL-reduct. Intuitively, $sel(I, S) \vee not\ sel(I, S) \leftarrow$ can be used to freely introduce sel-literals, provided no other rules prohibit this, e.g., a constraint $\leftarrow sel(x, s)$ makes sure no answer set contains $sel(x, s)$. We will call a predicate f *free* if $f(X, Y) \vee not\ f(X, Y) \leftarrow$ or $f(X) \vee not\ f(X) \leftarrow$ is in the program, or is silently assumed to be in it, for a binary or unary f respectively. Similarly, a ground literal l is free if we have $l \vee not\ l \leftarrow$.

Open answer sets are a generalization of the k-belief sets in [12]. A k-belief set of a program P is a pair $\langle k, B\rangle$ where k is a nonnegative integer and B is an answer set of P_k, which is the grounding of P with its own constants and k new ones. Obviously, every k-belief set is an open answer set; the opposite is false as we may have infinite universes and, consequently, infinite open answer sets while k-belief sets are finite. Since reasoning, e.g., satisfiability checking, is undecidable under the k-belief semantics[25], reasoning under the open answer set semantics is too.

3 Conceptual Logic Programs

Since *Open Answer Set Programming* is, in general, undecidable, we seek to restrict the structure of DLPs to regain decidability while retaining enough expressiveness for solving practical problems. An important indication of decidability is the *tree-model property*, e.g., in modal logics[30], or its generalization, the *forest-model property*, as in DLs with individuals[18].

A program P has the forest-model property if the following holds: if P has an answer set that satisfies a unary predicate p, then P has an answer set with a forest shape that satisfies p in a root of a tree in this forest. E.g., consider the program P_3 representing the knowledge that a company can be trusted for doing business with if it has the ISO 9000 quality certificate and at least two different trustworthy companies are doing business with it:

$$\begin{aligned} trust(C) &\leftarrow t\_bus(C, C_1), t\_bus(C, C_2), C_1 \neq C_2, qual(C, iso9000) \\ &\leftarrow t\_bus(C, D), not\ trust(D) \end{aligned}$$

with t_bus and $qual$ free predicates, and $iso9000$ a constant. An answer set[3] of P_3, e.g., $M = \{trust(x_1), t\_bus(x_1, x_2), t\_bus(x_1, x_3), qual(x_1, iso9000), trust(x_2), \ldots\}$, is such that for every trusted company x_i in M, i.e. $trust(x_i) \in M$, there must be $t\_bus(x_i, x_j)$, $t\_bus(x_i, x_k)$ and $trust(x_j)$, $trust(x_k)$ with $x_j \neq x_k$; additionally, every trusted company has the $iso9000$ quality label. This particular answer set has a forest shape, as can be seen from Figure 1: we call it a *forest-model*. The forest in

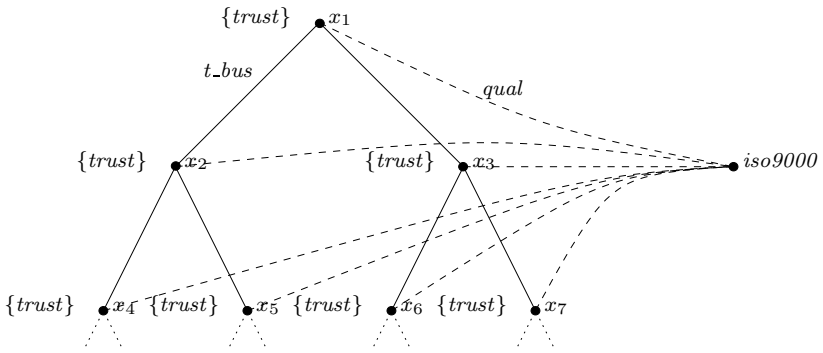

Fig. 1. Forest-Model

Figure 1 consists of two trees, one with root x_1 and one, a single node tree, with root $iso9000$. The labels of a node x in a tree, e.g., $\{trust\}$ for x_2, encode which literals are in the corresponding answer set, e.g. $trust(x_2) \in M$, while the labeled edges indicate relations between domain elements. The dashed arrows, describing relations between anonymous domain elements $x \in \mathcal{H} \setminus \mathcal{H}_P$, and constants, appear to be violating the forest structure; their labels can, however, be stored in the label of the starting node, e.g., $qual(x_2, iso9000)$ can be kept in the label of x_2 as $qual^{iso9000}$. Since there are only a finite number of constants, the number of different labels in a forest would still be finite. It is clear that M satisfies the predicate $trust$ in the root of a tree.

A particular class of programs with this forest-model property are *Conceptual Logic Programs* (CLPs).

Definition 1. *A CLP is a DLP such that a rule is of one of the following types:*

- ***free rules*** $l \vee not\ l \leftarrow$ *for a literal* l*, which allow for the free addition of the literal* l*, if not prohibited by other rules,*
- ***unary rules***[4] $a(s) \leftarrow \beta(s), \cup_m \gamma_m(s, t_m), \cup_m \delta_m(t_m), \cup_{i \neq j} t_i \neq t_j$*, such that, if* $\gamma_m \neq \emptyset$ *then* $\gamma_m^+ \neq \emptyset$*, and, in case* t_m *is a variable: if* $\delta_m \neq \emptyset$ *then* $\gamma_m \neq \emptyset$*,*
- ***binary rules*** $f(s, t) \leftarrow \beta(s), \gamma(s, t), \delta(t)$ *with* $\gamma^+ \neq \emptyset$ *if* t *is a variable,*
- ***constraints*** $\leftarrow a(s)$.

[3] The universe $\mathcal{H}$ can be deduced from this answer set.

[4] We will write unary rules, for compactness, as $a(s) \leftarrow \beta(s), \gamma_m(s, t_m), \delta_m(t_m), t_i \neq t_j$, with variables assumed to be pairwise different.

where i and j are within the range of m.

P_1, P_2, and P_3 are examples of CLPs. CLPs are designed to ensure the forest-model property. E.g., a rule $q(X) \leftarrow not\ f(X, Y), \neg q(Y)$ is not a CLP rule since, if $\neg q$ is free, $\{q(x), \neg q(y)\}$ is an answer set that cannot be transformed into a tree due to the lack of a connection between nodes x and y. The same argument applies to rules of the form $q(X) \leftarrow \neg q(Y)$. One may have, however, a rule $q(X) \leftarrow \neg q(a)$ for a constant a, since an answer set $\{q(x), \neg q(a)\}$ consists of two trees, with roots x and a respectively.

A rule $f(X, Y) \leftarrow v(X)$ is not allowed since it may enforce f-connections that break the tree-structure. On the other hand, $f(X, a) \leftarrow v(X)$ is allowed, as it only connects nodes x and the constant a. Note that more general rules than the ones in Definition 1 can be easily obtained by unfolding atoms in the bodies, resulting in rules with a tree structure. A complicated constraint $\leftarrow \beta$ is equivalent to the unary rule $a(s) \leftarrow \beta$ and the simple constraint $\leftarrow a(s)$. The idea of ensuring such connectedness of models in order to have desirable properties, like decidability, is similar to the motivation behind the *guarded fragment* of predicate logic[3].

Theorem 1. *Conceptual logic programs have the forest-model property.*

Forest-models of a CLP consist of at most $c + 1$ trees, with c the number of constants in the program. Each constant is the root of a tree, and an extra tree may be needed if a predicate can only be satisfied by an anonymous element, which will be the root of this tree.

Those trees may be infinite, but have bounded branching. For every label of a node x containing a predicate p, we have that $p(x)$ is in the forest-model, such that there must be some rule $p(x) \leftarrow \beta^+(x), \gamma_m^+(x, y_m), \delta_m^+(y_m)$ with a true body (if there were no such rule there would be no reason to include $p(x)$ in the forest-model, violating the minimality of answer sets). Thus, intuitively, in order to make p true in x, one needs to introduce at most $|\{y_m\}|$ successor nodes[5]. Since the size of the label at x is, roughly, bounded by the number of predicates in the program, this introduction of new successors of x only needs to occur a bounded number of times, resulting in the bounded branching.

In [15], decidability of satisfiability checking was shown by a reduction to two-way alternating tree-automata[31]. Since the CLPs in this paper also contain constants the automata reduction is not directly applicable. Moreover, while automata provide an elegant characterization, few implementations are available.

We slightly restrict CLPs, resulting in *local CLPs*, such that satisfiability checking can be reduced to normal, finite ASP, and, consequently, performed by existing answer set solvers such as DLV[21] and Smodels[27].

We first indicate how infinite forest-models can be turned into finite answer sets: cut every path in the forest from the moment there are duplicate labels and copy the connections of the first node in such a duplicate pair to the second node of the pair. Intuitively, when we reach a node that is in a state we already encountered, we proceed as that previous state, instead of going further down the tree. This cutting is similar to

[5] This bound can be easily tightened, e.g., if y_m is a constant there is no need for a successor y_m, since constants are treated as roots of their own tree.

the blocking technique for DL tableaux[4], but the minimality of answer sets makes it non-trivial and only valid for local CLPs, as we indicate below. Considering the forest-model in Figure 1, we can cut everything below x_2 and x_3 since they have the same label as x_1. Furthermore, since $t\_bus(x_1, x_2)$, $t\_bus(x_1, x_3)$, and $qual(x_1, iso9000)$, we have $t\_bus(x_i, x_2)$, $t\_bus(x_i, x_3)$, and $qual(x_i, iso9000)$ for $i = 2$ and $i = 3$, resulting in the answer set depicted in Figure 2.

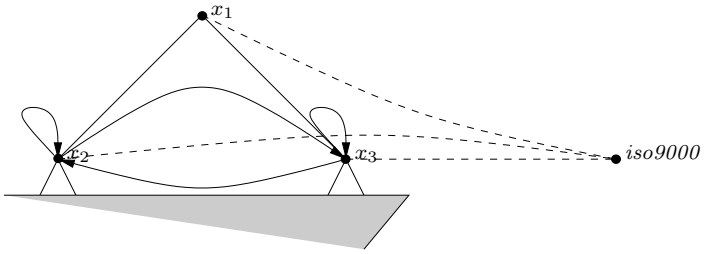

Fig. 2. Bounded Finite Model

This *cutting* is not possible for arbitrary CLPs. E.g., $a(X) \leftarrow f(X, Y), a(Y)$ and $a(X) \leftarrow b(X)$ with b and f free predicates. A possible forest-model of this small program is $\{a(x), f(x, y), a(y), f(y, z), b(z), a(z)\}$ with a tree $\{x \rightarrow y \rightarrow z\}$. Since x and y have the same label we cut at y, however, in the resulting answer set $a(x)$ is not motivated, as $b(z)$ is no longer present. The result of cutting is thus not minimal. Local CLPs solve this by making sure that a literal $a(x)$ is always motivated by x itself, successors y of x, or constants, such that, upon cutting, no motivating literals for literals higher up in the tree are cut away. Formally, local[6] CLPs are CLPs where rules $a(s) \leftarrow \alpha(s), \gamma_m(s, t_m), \beta_m(t_m), t_i \neq t_j$ and $f(s, t) \leftarrow \alpha(s), \gamma(s, t), \beta(t)$ are such that for every $b \in \beta^+_{(m)}$, either $b(t_{(m)}) \vee not\ b(t_{(m)}) \leftarrow\in P$ or for all rules $r : b(s) \leftarrow \text{body}(r)$, $\text{body}(r)^+ = \emptyset$. The programs P_1, P_2, and P_3 are local CLPs.

Every infinite forest-model of a local CLP can thus be made into a finite answer set, and moreover, we can put a bound, depending only on the program, on the number of domain elements that are needed for the finite version. Since there are only a finite number of labels m, every path of length longer than m will contain a duplicate label. The branching of every tree in a forest-model is also bounded, say by n, and there is a bounded number of trees in the forest-model ($c + 1$ for c the number of constants in the program), such that the number of nodes in an answer set that resulted from cutting is bounded by some k_P for a local CLP P. We can then reduce satisfiability checking w.r.t. a local CLP P to normal ASP by introducing at least k_P constants.

[6] The conditions for local are too strict, as is shown in [14], in the sense that there are CLPs that are not local but for which the infinite answer sets can still be made finite. However, since local CLPs are a syntactical restriction of CLPs, locality is a sufficient condition that is easy to check.

Theorem 2. *Let P be a local CLP. $p \in upreds(P)$ is satisfiable w.r.t. P iff there is an answer set M of $\psi(P)$ containing a $p(x_i)$, $1 \leq i \leq k_P$, where $\psi(P) = P \cup \{cte(x_i) \leftarrow \mid 1 \leq i \leq k_P\}$.*

In the non-trivial "only if" direction, a forest-model will be transformed into an answer set containing less than k_P domain elements by the cutting technique described above, which in turn will be mapped to the constants of $\psi(P)$.

4 Semantic Web Reasoning with Conceptual Logic Programs

Description Logics[4] play an important role in the deployment of the Semantic Web, as they provide the formal semantics of (part of) ontology languages such as OWL[5]. Using *concept* and *role names* as basic building blocks, *terminological* and *role axioms* in such DLs define subset relations between complex *concept* and *role expressions* respectively.

The semantics of DLs is given by interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty domain and $\cdot^{\mathcal{I}}$ is an interpretation function. $\mathcal{ALCHOQ}(\sqcup, \sqcap)$[7] is a particular DL with syntax and semantics as in Table 1; concept names A and individuals $\{o\}$ are the base concept expressions, P is a role name, establishing the base role expression, D and E are arbitrary concept expressions, and R and S are arbitrary role expressions.

Table 1. Syntax and Semantics $\mathcal{ALCHOQ}(\sqcup, \sqcap)$

concept names	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role names	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
individuals	$\{o\}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}, \lvert\{o\}^{\mathcal{I}}\rvert = 1$
conjunction of concepts	$(D \sqcap E)^{\mathcal{I}} = D^{\mathcal{I}} \cap E^{\mathcal{I}}$
disjunction of concepts	$(D \sqcup E)^{\mathcal{I}} = D^{\mathcal{I}} \cup E^{\mathcal{I}}$
conjunction of roles	$(R \sqcap S)^{\mathcal{I}} = R^{\mathcal{I}} \cap S^{\mathcal{I}}$
disjunction of roles	$(R \sqcup S)^{\mathcal{I}} = R^{\mathcal{I}} \cup S^{\mathcal{I}}$
existential restriction	$(\exists R.D)^{\mathcal{I}} = \{x \mid \exists y : (x,y) \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\}$
universal restriction	$(\forall R.D)^{\mathcal{I}} = \{x \mid \forall y : (x,y) \in R^{\mathcal{I}} \Rightarrow y \in D^{\mathcal{I}}\}$
qualified number restriction	$(\leq n\ R.D)^{\mathcal{I}} = \{x \mid \#\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\} \leq n\}$
	$(\geq n\ R.D)^{\mathcal{I}} = \{x \mid \#\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\} \geq n\}$

The *unique name assumption* - if $\{o_1\} \neq \{o_2\}$ then $\{o_1\}^{\mathcal{I}} \neq \{o_2\}^{\mathcal{I}}$ - ensures that different individuals are interpreted as different domain elements[8]. For concept expressions D and E, *terminological axioms* $D \sqsubseteq E$ are satisfied by an interpretation

[7] DLs are named according to their constructs: $\mathcal{AL}$ is the basic DL[26], and $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ adds negation of concept expressions ($\mathcal{C}$), role hierarchies ($\mathcal{H}$), individuals (or nominals) ($\mathcal{O}$), qualified number restrictions ($\mathcal{Q}$), and conjunction ($\sqcap$) and disjunction ($\sqcup$) of roles.

[8] Note that OWL does not make the unique name assumption, but one may enforce it using the *AllDifferent* construct.

$\mathcal{I}$ if $D^{\mathcal{I}} \subseteq E^{\mathcal{I}}$. Role axioms $R \sqsubseteq S$ are interpreted similarly. An axiom $X \equiv Y$ stands for $X \sqsubseteq Y$ and $Y \sqsubseteq X$. A *knowledge base* Σ is a set of terminological and role axioms; $\mathcal{I}$ is a *model* of Σ if $\mathcal{I}$ satisfies every axiom in Σ. A concept expression C is *satisfiable* w.r.t. Σ if there exists a model $\mathcal{I}$ of Σ such that $C^{\mathcal{I}} \neq \emptyset$.

As an example, the human resources department may have an ontology specifying the company's structure: (a) $Personnel$ consists of $Management$, $Workers$ and $john$, (b) $john$ is the boss of some manager, and (c) managers only take orders from other managers and are the boss of at least three $Workers$. This corresponds to the following $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base Σ_1:

$$\begin{aligned} Personnel &\equiv Management \sqcup Workers \sqcup \{john\} \\ \{john\} &\sqsubseteq \exists boss.Management \\ Management &\sqsubseteq (\forall t\_orders.Management) \sqcap (\geq\ 3\ boss.Workers) \end{aligned}$$

A model of this knowledge base is $\mathcal{I} = (\{j, w_1, w_2, w_3, m\}, \cdot^{\mathcal{I}})$, with $\cdot^{\mathcal{I}}$ defined by $Workers^{\mathcal{I}} = \{w_1, w_2, w_3\}$, $Management^{\mathcal{I}} = \{m\}$, $\{john\}^{\mathcal{I}} = \{j\}$, $Personnel^{\mathcal{I}} = \{j, w_1, w_2, w_3, m\}$, $boss^{\mathcal{I}} = \{(j, m), (m, w_1), (m, w_2), (m, w_3)\}$, and $t\_orders^{\mathcal{I}} = \emptyset$.

We can rewrite Σ_1 as an equivalent CLP P_4. The axioms in Σ_1 correspond to the constraints

$$\begin{aligned} &\leftarrow Personnel(X), not\ (Management \sqcup Workers \sqcup \{john\})(X) \\ &\leftarrow (Management \sqcup Workers \sqcup \{john\})(X), not\ Personnel(X) \\ &\leftarrow \{john\}(X), not\ (\exists boss.Management)(X) \\ &\leftarrow Management(X), not\ ((\forall t\_orders.Management) \sqcap (\geq\ 3\ boss.Workers))(X) \end{aligned}$$

in P_4, where the concept expressions are used as predicates, and indicating, in case of the first constraint, that if the answer set contains some $Personnel(x)$ then it must also contain $(Management \sqcup Workers \sqcup \{john\})(x)$. Those constraints are the kernel of the translation; we still need, however, to simulate the DLs semantics by rules that define the different DL constructs.

The predicate $(Management \sqcup Workers \sqcup \{john\})$ is defined by rules

$$\begin{aligned} (Management \sqcup Workers \sqcup \{john\})(X) &\leftarrow Management(X) \\ (Management \sqcup Workers \sqcup \{john\})(X) &\leftarrow Workers(X) \\ (Management \sqcup Workers \sqcup \{john\})(X) &\leftarrow \{john\}(X) \end{aligned}$$

and thus, by minimality of answer sets, if $(Management \sqcup Workers \sqcup \{john\})(x)$, there must either be a $Management(x)$, a $Workers(x)$, or a $\{john\}(x)$. The other way around, if one has a $Management(x)$, a $Workers(x)$, or a $\{john\}(x)$, one must have, since answer sets are models, $(Management \sqcup Workers \sqcup \{john\})(x)$. This behavior is exactly what is required by the $\sqcup$-construct.

The predicate $(\exists boss.Management)$ is defined by $(\exists boss.Management)(X) \leftarrow boss(X, Y), Management(Y)$, such that, if $(\exists boss.Management)(x)$ is in the answer set, there must be, by minimality, a y such that $boss(x, y)$ and $Management(y)$ are in the answer set and vice versa.

The conjunction predicate $((\forall t\_orders.Management) \sqcap (\geq\ 3\ boss.Workers))$ is defined by

$$((\forall t\_orders.Management) \sqcap (\geq\ 3\ boss.Workers))(X) \leftarrow (\forall t\_orders.Management)(X), (\geq\ 3\ boss.Workers)(X)$$

and the body predicates by the rules

$$\begin{aligned}(\forall t\_orders.Management)(X) &\leftarrow not\ \exists t\_orders.\neg Management(X)\\ (\geq\ 3\ boss.Workers)(X) &\leftarrow boss(X, Y_1), boss(X, Y_2), boss(X, Y_3),\\ &\quad Workers(Y_1), Workers(Y_2), Workers(Y_3),\\ &\quad Y_1 \neq Y_2, Y_2 \neq Y_3, Y_1 \neq Y_3\end{aligned}$$

and

$$\begin{aligned}\exists t\_orders.\neg Management(X) &\leftarrow t\_orders(X, Y), (\neg Management)(Y)\\ (\neg Management)(X) &\leftarrow not\ Management(X)\end{aligned}$$

Finally, we need to introduce free rules for all concept and role names. Intuitively, concept names and roles names are types and thus contain some instances or not.

$$\begin{aligned}Workers(X) \vee not\ Workers(X) &\leftarrow\\ Personnel(X) \vee not\ Personnel(X) &\leftarrow\\ Management(X) \vee not\ Management(X) &\leftarrow\\ boss(X, Y) \vee not\ boss(X, Y) &\leftarrow\\ t\_orders(X, Y) \vee not\ t\_orders(X, Y) &\leftarrow\end{aligned}$$

The individual $\{john\}$ is taken care of by introducing a constant $john$ in the program with the rule $\{john\}(john) \leftarrow$. The only possible value of X in a $\{john\}(X)$ is then $john$.

The DL model $\mathcal{I}$ corresponds to the open answer set $(\mathcal{H}, M)$ with $\mathcal{H} = (\Delta^{\mathcal{I}} \setminus \{j\}) \cup \{john\}$ and $M = \{C(x) \mid C \in upreds(P_4), x \in C^{\mathcal{I}}\} \cup \{R(x, y) \mid R \in bpreds(P_4), (x, y) \in R^{\mathcal{I}}\}$, with a slight abuse of notation, i.e. using C and R as predicates and DL expressions. Formally, we define the *closure* $clos(C, \Sigma)$ of a concept expression C and a knowledge base Σ as the smallest set satisfying the following conditions:

- for every concept (role) expression D (R) in $\{C\} \cup \Sigma$ we have $D(R) \in clos(C, \Sigma)$,
- for every D in $clos(C, \Sigma)$, we distinguish the following cases:
  - $D = \neg D_1 \quad \Rightarrow D_1 \in clos(C, \Sigma)$
  - $D = D_1 \sqcup D_2 \quad \Rightarrow \{D_1, D_2\} \subseteq clos(C, \Sigma)$
  - $D = D_1 \sqcap D_2 \quad \Rightarrow \{D_1, D_2\} \subseteq clos(C, \Sigma)$
  - $D = \exists R.D_1 \quad \Rightarrow \{R, D_1\} \subseteq clos(C, \Sigma)$
  - $D = \forall R.D_1 \quad \Rightarrow \{D_1, \exists R.\neg D_1\} \subseteq clos(C, \Sigma)$
  - $D = (\leq n\ Q.D_1) \Rightarrow \{(\geq n+1\ Q.D_1)\} \subseteq clos(C, \Sigma)$
  - $D = (\geq n\ Q.D_1) \Rightarrow \{Q, D_1\} \subseteq clos(C, \Sigma)$
- for $R \sqcup S \in clos(C, \Sigma)$, $\{R, S\} \subseteq clos(C, \Sigma)$,
- for $R \sqcap S \in clos(C, \Sigma)$, $\{R, S\} \subseteq clos(C, \Sigma)$.

The CLP $\Phi(C, \Sigma)$ that simulates satisfiability checking of C w.r.t. Σ is then constructed by introducing for concept names A, role names P, and individuals $\{o\}$ in $clos(C, \Sigma)$, rules $A(X) \vee not\ A(X) \leftarrow$, $P(X, Y) \vee not\ P(X, Y) \leftarrow$, and facts $\{o\}(o) \leftarrow$. For every other construct $B \in clos(C, \Sigma)$, we introduce, depending on the particular construct, a rule with B in the head as in Table 2.

Table 2. CLP Translation $\Phi(C, \Sigma)$

$\neg D(X) \leftarrow not\ D(X)$	$D \sqcap E(X) \leftarrow D(X), E(X)$
$D \sqcup E(X) \leftarrow D(X)$	$D \sqcup E(X) \leftarrow E(X)$
$\exists R.D(X) \leftarrow R(X,Y), D(Y)$	$\forall R.D(X) \leftarrow not\ \exists R.\neg D(X)$
$R \sqcup S(X,Y) \leftarrow R(X,Y)$	$R \sqcap S(X,Y) \leftarrow R(X,Y), S(X,Y)$
$R \sqcup S(X,Y) \leftarrow S(X,Y)$	$(\leq n\ R.D)(X) \leftarrow not\ (\geq n+1\ R.D)(X)$
$(\geq n\ R.D)(X) \leftarrow R(X,Y_1), \ldots, R(X,Y_n), D(Y_1), \ldots, D(Y_n), Y_1 \neq Y_2, \ldots$	

This completes the simulation of $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ using CLP.

Theorem 3. *An $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ concept expression C is satisfiable w.r.t. a knowledge base Σ iff C is satisfiable w.r.t. $\Phi(C, \Sigma)$.*

Proof Sketch. For the "only if" direction, take C satisfiable w.r.t. Σ, i.e. there exists a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $C^{\mathcal{I}} \neq \emptyset$. We rename the element $x \in \{o\}^{\mathcal{I}}$ from $\Delta^{\mathcal{I}}$ by o, which is possible by the unique name assumption. We then construct the answer set $(\mathcal{H}, M)$ with $\mathcal{H} = \Delta^{\mathcal{I}}$ and $M = \{C(x) \mid x \in C^{\mathcal{I}}, C \in clos(C, \Sigma)\} \cup \{R(x, y) \mid (x, y) \in R^{\mathcal{I}}, R \in clos(C, \Sigma)\}$. One can show that $(\mathcal{H}, M)$ is an answer set of $\Phi(C, \Sigma)$.

For the "if" direction, we have an open answer set $(\mathcal{H}, M)$ that satisfies C, i.e. $C(x) \in M$ for some $x \in \mathcal{H}$. Define an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, with $\Delta^{\mathcal{I}} = \mathcal{H}$, and $A^{\mathcal{I}} = \{y \mid A(y) \in M\}$, for concept names A, $P^{\mathcal{I}} = \{(y, z) \mid P(y, z) \in M\}$, for role names P, and $\{o\}^{\mathcal{I}} = \{o\}$, for $o \in \mathcal{H}_{\Phi(C,\Sigma)}$. $\mathcal{I}$ is defined on concept expressions and role expressions as in Table 1, and we can show that $\mathcal{I}$ is a model of Σ such that $C^{\mathcal{I}} \neq \emptyset$. □

Note that, in general, the resulting CLP $\Phi(C, \Sigma)$ is not local, e.g., a DL expression $\exists R.(A \sqcap B)$ is translated as the rules $\exists R.(A \sqcap B)(X) \leftarrow R(X, Y), A \sqcap B(Y)$ and $A \sqcap B(X) \leftarrow A(X), B(X)$, such that there is a positive $A \sqcap B$ atom that is not free in a body and there is a rule with $A \sqcap B$ in the head and a body that has a non-empty positive part. $\Phi(C, \Sigma)$ has, however, the convenient property that it is *positively acyclic*, i.e. recursion only occurs through negative (with naf) literals; for more details, see [14]. It is sufficient to note that the body of a rule in $\Phi(C, \Sigma)$ is structurally "smaller" than the head, e.g., $A \sqcap B$ is smaller than $\exists R.A \sqcap B$. This permits us to replace the rule with $\exists R.A \sqcap B$ in the head by the two rules $\exists R.(A \sqcap B)(X) \leftarrow R(X, Y), not\ (A \sqcap B)'(Y)$; $(A \sqcap B)'(X) \leftarrow not\ (A \sqcap B)(X)$; i.e. we negate $A \sqcap B(Y)$ twice. The resulting CLP is now local.

Such a procedure does not work for arbitrary CLPs, e.g., we have that $(\{x\}, \{l(x)\})$ is not an open answer set of the rule $l(X) \leftarrow l(X)$, since, although it is a model of

$l(x) \leftarrow l(x)$, it is not a minimal model - the empty set is. Transforming the rule, however, by doubly negating the body yields $l(X) \leftarrow not\ l'(X)$ and $l'(X) \leftarrow not\ l(X)$, which does have $(\{x\}, \{l(x)\}$ as an answer set since the GL-reduct contains only the rule $l(x) \leftarrow$.

The $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ simulation shows the feasibility of Semantic Web reasoning with CLPs, as $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ is an expressive DL closely related to the DL $\mathcal{SHOIQ}(\mathbf{D})$, i.e. $\mathcal{SHOQ}(\mathbf{D})$[18] with support for inverted roles, and $\mathcal{SHOIQ}(\mathbf{D})$ is the DL corresponding to the ontology language OWL DL[5]. $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ differs from the DL $\mathcal{SHOIQ}(\mathbf{D})$ by its lack of inverted roles, data types ($\mathbf{D}$) and transitivity of roles (which distinguish $\mathcal{S}$ from $\mathcal{ALC}$); it adds the role constructs $\sqcup$ and $\sqcap$ though.

Since CLP, as a logic programming paradigm, is a natural framework for expressing rules, it can be used to represent and reason with both ontological and rule-based knowledge. Additionally, CLP enables nonmonotonic reasoning on the Semantic Web, identified in [7] as one of the requirements on a logic for reasoning on the Web.

Translating existing DL ontologies to CLP or devising new ontologies that need only DL-like constructs with CLP is not always a good idea. As one sees from the above simulation, the CLP version of a DL ontology produces a lot of overhead rules, specifying the implicit DL semantics. As a result, the translated CLP is likely to be less compact than the original DL knowledge base. However, two remarks are in order here. Firstly, CLPs could nevertheless prove useful as an underlying implementation mechanism for uniform reasoning with both DL ontologies and CLP rules: they ensure a decidable environment for making inferences. Secondly, not all common knowledge can be elegantly represented by DLs; some useful constructs cannot be represented at all. We highlight three advantages of using CLPs for representing knowledge:

Closed Domain Reasoning. Using CLPs, we can explicitly close the domain, i.e. only allow reasoning with constants. Indeed, one can, as in [12], simply add the rules $H(a) \leftarrow$ for every constant a, and a constraint $\leftarrow not\ H(X)$ such that all domain elements must be constants. A similar intervention, restricting the reasoning to individuals, is impossible within standard DLs[9] and was one of the arguments to extend DLs with nonmonotonic tools[9].

Generalized Number Restrictions. The translation of DL ontologies tend to produce some overhead, however, CLPs are more articulate than DLs in other aspects. E.g., representing the knowledge that a team must at least consist of a technical expert, a secretary, and a team leader, where the leader and the technical expert are not the same, can be done by $team(X) \leftarrow member(X, Y_1), tech(Y_1), member(X, Y_2), secret(Y_2), leader(X, Y_3), Y_1 \neq Y_3$. Note that this definition of a team does not exclude non-listed members to be part of the team. Moreover, in the presence of other rules with $team$ in the head, a team may be qualified by one of those rules. E.g., including a fact $team(007)$, would qualify 007 as a team, regardless of its members. Representing such *generalized number restrictions* using DLs would be significantly harder while arguably less succinct.

Query Containment, Consistency, and Disjointness. Those three query problems were identified in [4] as important for ontology reasoning. Query containment is the problem of deciding whether for every database D satisfying an ontology, the result

[9] One could enforce closed domain reasoning in DLs by working internally with CLPs.

of a query Q_1 to D is contained in the result of Q_2 to D. Instead of the usual conjunctive Datalog queries, we can use CLPs to represent both queries and ontology. E.g., a query $Q_1(X) \leftarrow Management(X)$ retrieves the managers and $Q_2(X) \leftarrow boss(X, Y_1), boss(X, Y_2), Y_1 \neq Y_2$ retrieves the persons that supervise more than two persons. Clearly, Q_1 is contained in Q_2 w.r.t. the ontology P_4 since, according to P_4, managers must supervise at least three workers.

Moreover, all three query problems can be reduced to satisfiability checking w.r.t. a CLP; intuitively, in the query containment case, one extends the ontology with a rule $r(X) \leftarrow Q_1(X), not\ Q_2(X)$, against which unsatisfiability of r is checked. More detail can be found in [14].

5 Related Work

There are basically two lines of research that try to reconcile Description Logics with Logic Programming. The approaches in [6, 13, 23, 2, 20, 28] simulate DLs with LP, possibly with a detour to FOL, while [8, 24, 10] attempt to unite the strengths of DLs and LP by letting them coexist and interact.

In [6], the simulation of a DL with acyclic axioms in *open logic programming* is shown. An open logic program is a program with possibly undefined predicates and a FOL-theory; the semantics is the completion semantics, which is only complete for a restrictive set of programs. The open-nes lies in the use of undefined predicates, which are comparable to free predicates with the difference that free predicates can be expressed within the CLP framework. More specifically, open logic programming simulates reasoning in the DL $\mathcal{ALCN}$, $\mathcal{N}$ indicating the use of unqualified number restrictions, where terminological axioms consist of non-recursive concept definitions; $\mathcal{ALCN}$ is a subclass of $\mathcal{ALCHOQ}(\sqcup, \sqcap)$.

[13] imposes restrictions on the occurrence of DL constructs in terminological axioms to enable a simulation using Horn clauses. E.g., axioms containing disjunction on the right hand side, as in $D \sqsubseteq C \sqcup D$, universal restriction on the left hand side, or existential restriction on the right hand side are prohibited since Horn clauses cannot represent them. Moreover, neither negation of concept expressions nor number restrictions can be represented. So-called *Description Logic Programs* are thus incapable of handling expressive DLs; however, [13]'s forte lies in the identification of a subclass of DLs that make efficient reasoning through LPs possible. [23] extends the work in [13], for it simulates non-recursive $\mathcal{ALC}$ ontologies with disjunctive deductive databases. Compared with, possibly recursive, $\mathcal{ALCHOQ}(\sqcup, \sqcap)$, those are still rather inexpressive.

In [2], the DL $\mathcal{ALCQI}$ is successfully translated into a DLP. However, to take into account infinite interpretations [2] presumes, for technical reasons, the existence of function symbols, which leads, in general, to undecidability of reasoning.

[20] and [28] simulate reasoning in DLs with a LP formalism by using an intermediate translation to first-order clauses. In [20], $\mathcal{SHIQ}^-$ knowledge bases, i.e. $\mathcal{SHIQ}$ knowledge bases with the requirement that roles S in $(\leq nS.C)$ have no subroles, are reduced to first-order formulas, on which basic superposition calculus is then applied.

The result is transformed into a function-free version which is translated to a disjunctive Datalog program.

[28] translates $\mathcal{ALCQI}$ concepts to first-order formulate, grounds them with a finite number of constants, and transforms the result to a logic program. One can use a finite number of constants by the finite-model property for $\mathcal{ALCQI}$-concept expressions; in the presence of terminological axioms this is no longer possible. The resulting program is, however, not declarative anymore such that its main contribution is that it provides an alternative reasoner for DLs, whereas CLPs can be used both for reasoning with DLs and for a direct and elegant expression of knowledge. Furthermore, CLPs are also interesting from a pure LP viewpoint since they constitute a decidable class of DLPs under the open answer set semantics.

Along the second line of research, an $\mathcal{AL}$-log[8] system consists of two subsystems: a DL knowledge base and a Datalog program, where in the latter variables may range over DL concept instances, thus obtaining a flow of information from the structural DL part to the relational Datalog part. This is extended in [24] for disjunctive Datalog and the $\mathcal{ALC}$ DL. A further generalization is attained in [10] where the particular DL can be the expressive $\mathcal{SHIF}(\mathbf{D})$, $\mathcal{F}$ stands for functional restrictions, or $\mathcal{SHOIN}(\mathbf{D})$. Moreover, the flow of information can go both ways.

Finally, a notable approach, which cannot be categorized in one of the two lines of research described above[10], is the SWRL[19] initiative. SWRL is a *Semantic Web Rule Language* and extends the syntax and semantics of OWL DL with unary/binary Datalog RuleML[1], i.e. Horn-like rules. This extension is undecidable[17] but lacks, nevertheless, interesting knowledge representation mechanisms such as negation as failure.

A reduction from query problems to (un)satisfiability problems for DLs may be found in [29].

6 Conclusions and Directions for Further Research

We extended ASP with open domains, defined CLPs to regain decidability, and reduced reasoning with CLPs to finite, closed, ASP. The simulation of an expressive fragment of the OWL DL ontology language, as well as additional LP mechanisms such as negation as failure and closed world reasoning, illustrates the relevance of CLPs for Semantic Web reasoning. We concluded with a description of related work.

We plan to further relax the restrictions on CLPs by working towards a graph-model property. A prototype implementation, using heuristics, is also envisaged.

References

1. The Rule Markup Initiative. http://www.ruleml.org.
2. G. Alsaç and C. Baral. Reasoning in Description Logics using Declarative Logic Programming. http://www.public.asu.edu/ guray/dlreasoning.pdf, 2002.
3. H. Andréka, I. Németi, and J. Van Benthem. Modal languages and bounded fragments of predicate logic. *J. of Philosophical Logic*, 27(3):217–274, 1998.

[10] Although it tends towards the coexisting approach.

4. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.
5. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference, 2004.
6. K. Van Belleghem, M. Denecker, and D. De Schreye. A Strong Correspondence between DLs and Open Logic Programming. In *Proc. of ICLP'97*, pages 346–360, 1997.
7. F. Bry and S. Schaffert. An Entailment Relation for Reasoning on the Web. In *Proc. of Rules and Rule Markup Languages for the Semantic Web*, LNCS, pages 17–34. Springer, 2003.
8. F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *J. of Intell. and Cooperative Information Systems*, 10:227–252, 1998.
9. F. M. Donini, D. Nardi, and R. Rosati. Description Logics of Minimal Knowledge and Negation as Failure. *ACM Trans. Comput. Logic*, 3(2):177–225, 2002.
10. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with DLs for the Semantic Web. In *Proc. of KR 2004*, pages 141–151, 2004.
11. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of ICLP'88*, pages 1070–1080, Cambridge, Massachusetts, 1988. MIT Press.
12. M. Gelfond and H. Przymusinska. Reasoning in Open Domains. In *Logic Programming and Non-Monotonic Reasoning*, pages 397–413. MIT Press, 1993.
13. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of WWW 2003*, pages 48–57, 2003.
14. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Decidable Open Answer Set Programming. Technical report, Vrije Universiteit Brussel, Dept. of Computer Science, 2004.
15. S. Heymans and D. Vermeir. Integrating Description Logics and Answer Set Programming. In *Proc. of PPSWR 2003*, number 2901 in LNCS, pages 146–159. Springer, 2003.
16. S. Heymans and D. Vermeir. Integrating Ontology Languages and Answer set Programming. In *Proc. of WebS'03*, pages 584–588. IEEE Computer Society, 2003.
17. I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*. ACM, 2004.
18. I. Horrocks and U. Sattler. Ontology Reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic. In *Proc. of IJCAI'01*, pages 199–204. Morgan Kaufmann, 2001.
19. I. Horrocks, P. F. Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule language Combining OWL and RuleML, May 2004.
20. U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. FZI-Report 1-8-11/03, Forschungszentrum Informatik (FZI), 2003.
21. N. Leone, W. Faber, and G. Pfeifer. DLV homepage. http://www.dbai.tuwien.ac.at/proj/dlv/.
22. V. Lifschitz. Answer Set Programming and Plan Generation. *AI*, 138(1-2):39–54, 2002.
23. B. Motik, R. Volz, and A. Maedche. Optimizing Query Answering in Description Logics using disjunctive deductive databases. In *Proc. of KRDB'03*, pages 39–50, 2003.
24. R. Rosati. Towards Expressive KR Systems Integrating Datalog and Description Logics: Preliminary Report. In *Proc. of DL'99*, pages 160–164, 1999.
25. J. Schlipf. Some Remarks on Computability and Open Domain Semantics. In *Proc. of the Worksh. on Struct. Complexity and Recursion-Theoretic Methods in Log. Prog.*, 1993.
26. M. Schmidt-Schaub and G. Smolka. Attributive Concept Descriptions with Complements. *Artif. Intell.*, 48(1):1–26, 1991.
27. P. Simons. Smodels homepage. http://www.tcs.hut.fi/Software/smodels/.
28. T. Swift. Deduction in Ontologies via Answer Set Programming. In Vladimir Lifschitz and Ilkka Niemelä, editors, *LPNMR*, volume 2923 of *LNCS*, pages 275–288. Springer, 2004.
29. S. Tessaris. Querying expressive DLs. In *Proc. of DL-2001*, 2001.
30. M. Y. Vardi. Why is Modal Logic so Robustly Decidable? Technical report, 1997.
31. M. Y. Vardi. Reasoning about the Past with Two-Way Automata. In *Proc. of ICALP '98*, pages 628–641. Springer, 1998.