Nonmonotonic Ontological and Rule-Based Reasoning with Extended 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 present *extended conceptual logic programs (ECLPs)*, for which reasoning is decidable and, moreover, can be reduced to finite answer set programming. ECLPs are useful to reason with both ontological and rule-based knowledge, which is illustrated by simulating reasoning in an expressive description logic (DL) equipped with DL-safe rules. Furthermore, ECLPs are more expressive in the sense that they enable nonmonotonic reasoning, a desirable feature in locally closed subareas of the Semantic Web.

1 Introduction

Reasoning with both ontological knowledge, in the form of a description logic (DL)[3] knowledge base, and rule-based knowledge has recently gained in interest in the Semantic Web community. The purpose of adding rules to ontological knowledge is to have additional expressiveness. E.g., [23] extends a DL knowledge base with *DL-safe rules*, i.e. Horn clauses where variables must appear in non-DL-atoms in the body of rules. DL-safe rules can, e.g., express triangular knowledge not expressible with DLs alone: $uncle(a, c) \leftarrow brother(a, b), parent(b, c)$.

DL-safe rules do not include the *negation as failure (naf)* operator, and as a consequence, do not cope well with incomplete or dynamically changing knowledge: like reasoning with DL, reasoning with DL knowledge bases and DL-safe rules is monotonic. However, nonmonotonic reasoning may be useful in applications that involve well-defined closed subareas of the Semantic Web, as illustrated in the following example. Assume a business is setting up its website for processing customer feedback. It decides to commit to an ontology $\mathcal{O}$ which defines that if there are no complaints for a product, it is a good product.

$$good\_product(X) \leftarrow not\ complaint(X)$$

The business has its own particular business rules, e.g. $i : invest(tps, \mathit{10K}) \leftarrow not\ good\_product(tps)$ saying that if its particular top selling product tps cannot be

[⋆] 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.

shown to be a good product, then the business has to invest 10K in tps. Finally, the business maintains a repository of dynamically changing knowledge, originating from user feedback collected on the site, e.g. at a certain time the repository contains $R_1 = \{complaint(tps) \leftarrow\}$ with a complaint for tps.

If the business wants to know whether to invest more in tps it needs to check $\mathcal{O} \cup \{i\} \cup R_1 \models invest(tps, 10K)$, i.e. whether the ontology, combined with its own business rules, and the information repository, demand for an investment or not.

One can use *extended conceptual logic programming (ECLP)* to express the above knowledge. Intuitively, any model of $\mathcal{O} \cup \{i\} \cup R_1$, must verify $complaint(tps)$, and thus $good\_product(X) \leftarrow not\ complaint(X)$ will not trigger and $good\_product(tps)$ will be false, which in turn, with rule i, allows to conclude that the business should indeed invest.

Evaluating the same query with an updated repository $R_2 = \{complaint(tps) \leftarrow, good\_product(tps) \leftarrow\}$ containing a survey result saying that tps is a good product, no matter what complaints of individual users there may be, leads to $\mathcal{O} \cup \{i\} \cup R_2 \not\models invest(tps, 10K)$, such that no further investments are necessary. Adding knowledge thus invalidates previous conclusions making reasoning nonmonotonic; similar scenarios can easily be imagined in any environment with dynamically changing knowledge.

In this paper, we formally introduce ECLP programs which consist of two (possibly empty) parts: a *conceptual logic program (CLP)* capable of expressing conceptual knowledge, as in e.g. DL knowledge bases, and an arbitrary *finite grounded program* which allows to relate constants/individuals in arbitrary ways, enabling e.g. the expression of triangular knowledge. More specifically, ECLPs can simulate reasoning in the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ equipped with DL-safe rules. Besides the advantage of uniform syntax and semantics that ECLPs have over DLs equipped with DL-safe rules[1], ECLPs are capable, as indicated above, of nonmonotonic reasoning as well.

Furthermore, we will show that reasoning with ECLPs can be reduced to finite answer set programming by virtue of the forest-model property and the bounded finite model property. The reduction to finite ASP makes reasoning with ECLPs amenable for existing answer set solvers such as DLV[21] or SMODELS[25].

The remainder of the paper is organized as follows. After recalling the open answer set semantics in Section 2, ECLPs are formally introduced in Section 3. Section 4 describes the simulation of an expressive class of DLs equipped with DL-safe rules. Section 5 highlights some related work while Section 6 contains conclusions and directions for further research. Due to space restrictions all proofs have been omitted; they can be found at `http://tinf2.vub.ac.be/~{}sheymans/tech/oasp-sw.ps.gz`.

2 Answer Set Programming with Open Domains

Answer set programming (ASP)[5] is a logic programming paradigm where knowledge is represented by programs and answer sets provide for the intended seman-

[1] SWRL[20] also combines ontologies and rules in one uniform syntax and semantics; reasoning with it is, however, undecidable.

tics of that knowledge. However, in certain cases ASP fails to capture the intention of the program. Take the program consisting of the rules $bad(X) \leftarrow not\ good(X)$ and $good(heather) \leftarrow$, where one is bad if not good and Heather is a good person. In ASP a program is grounded with the constants in the program, resulting in $bad(heather) \leftarrow not\ good(heather)$ and $good(heather) \leftarrow$, after which the unique answer set $\{good(heather\}$ can be calculated. One would thus wrongfully conclude that there can never be bad individuals. In [17], this was solved by considering *open domains*, i.e. the program may be grounded with any superset of the present constants: grounding with a universe $\{x, heather\}$ yields $bad(heather) \leftarrow not\ good(heather)$; $bad(x) \leftarrow not\ good(x)$ and $good(heather) \leftarrow$, which has an answer set $\{bad(x), good(heather)\}$, correctly capturing the intended meaning of the program.

We briefly recall the open answer set semantics from [17]. We call individual names *constants* and write them as lowercase letters, *variables* will be denoted with uppercase letters. Variables and constants are *terms*. *Atoms* are of the form $a(t)$ or $f(t_1, t_2)$, with a a unary predicate, f a binary predicate, and t, t_1 and t_2 terms. A *literal* is an atom or an atom preceded by $\neg$. An *extended literal* is a literal l or a *naf-literal* $not\ l$ where l is a literal. We will often denote a set of unary extended literals $\{a_1(s), \ldots, a_n(s)\}$, ranging over a common term s, as $\alpha(s)$ with $\alpha = \{a_1, \ldots, a_n\}$. A set of binary extended literals can be similarly denoted as $\alpha(s, t)$. The positive part of a set of extended literals β is $\beta^+ = \{l \mid l \in \beta, l \text{ literal}\}$, the negative part is $\beta^- = \{l \mid not\ l \in \beta\}$. Furthermore, we assume the existence of a binary predicate $\neq$, with the usual interpretation.

A *disjunctive logic program* (DLP) is a finite set of rules $r : \alpha \leftarrow \beta$ where α and β are finite sets of extended literals and $|\alpha^+| \leq 1$. If $\alpha = \emptyset$, the rule is called a *constraint*. The set α is the *head* of the rule r, denoted $\text{head}(r)$, while β is called the *body*, denoted $\text{body}(r)$. As usual, atoms, (extended) literals, rules, and programs that do not contain variables are *ground*. For a set X of literals, $\neg X = \{\neg l \mid l \in X\}$, where, by definition, $\neg\neg a \equiv a$. A set of ground literals X is *consistent* if $X \cap \neg X = \emptyset$.

For a DLP P, let $\mathcal{H}_P$ be the constants in P and $vars(P)$ its variables. A (possibly infinite) non-empty set of constants $\mathcal{H}$ such that $\mathcal{H}_P \subseteq \mathcal{H}$, is called a *universe* for P. We call $P_{\mathcal{H}}$ the *grounded program* obtained from P by substituting every variable in P by every possible constant in $\mathcal{H}$. Let $\mathcal{L}_P$ be the set of literals that can be formed from a grounded program P, $preds(P)$ are the predicates[2] in P, and $upreds(P)$ and $bpreds(P)$ the unary and binary predicates respectively.

An *interpretation* I of a grounded P is any consistent subset of $\mathcal{L}_P$. For a ground literal l, we write $I \models l$, if $l \in I$, which extends to $I \models not\ l$ if $I \not\models l$, and, for a set of ground extended literals X, $I \models X$ if $I \models x$ for every $x \in X$. A ground rule $r : \alpha \leftarrow \beta$ is *satisfied* w.r.t. I, denoted $I \models r$, if $I \models l$ for some $l \in \alpha$ whenever $I \models \beta$, i.e. r is *applied* whenever it is *applicable*. A ground constraint $\leftarrow \beta$ is satisfied w.r.t. I if $I \not\models \beta$. For a *simple* grounded program P (i.e. a program without *not*), I is a *model* of P if I satisfies every rule in P; it is an *answer set* of P if it is a subset minimal model of P. For grounded programs P containing *not*, the *GL-reduct*[13] w.r.t. I is defined as P^I, where P^I contains $\alpha^+ \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in P, $\beta^- \cap I = \emptyset$ and $\alpha^- \subseteq I$. I is an *answer set* of a grounded P if I is an answer set of P^I. An *open interpretation* of P

[2] When speaking of predicates, also the (classically) negated predicates are assumed.

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 then an open interpretation $(\mathcal{H}, M)$ with M an answer set of $P_{\mathcal{H}}$. In the following, we will usually omit the "open" qualifier. We express the motivation of a literal in an answer set formally by means of the operator T that computes the closure of a set of literals w.r.t. a GL-reduct. For a DLP P and an interpretation $(\mathcal{H}, M)$ of P, $T_{P^M_{\mathcal{H}}} : \mathcal{L}_{P^M_{\mathcal{H}}} \to \mathcal{L}_{P^M_{\mathcal{H}}}$ is defined as[3] $T(B) = B \cup \{a | a \leftarrow \beta \in P^M_{\mathcal{H}} \wedge \beta \subseteq B\}$. Additionally, we have $T^0(B) = B$, and $T^{n+1}(B) = T(T^n(B))$. More detail than the T-operator is provided by the *support* of a literal a in an answer set $(\mathcal{H}, M)$, which explicitly indicates the literals that support the presence of a in the answer set. For the least n such that $a \in T^n$, we inductively define the support $S^k(a)$ on a certain level $1 \leq k \leq n$ as $S^n(a) = \{a\}$ and $S^k(a) = \{\beta \mid b \leftarrow \beta \in P^M_{\mathcal{H}}, \beta \subseteq T^k, b \in S^{k+1}(a)\}$, $1 \leq k < n$. A support for a is then $S(a) = \cup_{k=1}^n S^k(a)$.

Take, for example, the program P with a rule $p(X) \vee not\ p(X) \leftarrow$. Grounding w.r.t. to a universe $\{x, y\}$ yields the program $P_{\{x,y\}}$ consisting of $p(x) \vee not\ p(x) \leftarrow$ and $p(y) \vee not\ p(y) \leftarrow$. We have that $\{p(x)\}$ is an answer set of $P_{\{x,y\}}$, since the GL-reduct is $p(x) \leftarrow$ which has only one minimal model: $\{p(x)\}$ itself. Thus $(\{x, y\}, \{p(x)\})$ is an answer set of P. Actually, a rule such as in P allows one to freely introduce p-literals (provided no other rules constrain this). We call a predicate p *free* if $p(X, Y) \vee not\ p(X, Y) \leftarrow$ or $p(X) \vee not\ p(X) \leftarrow$ is in the program, for a binary or unary p respectively. Similarly, a ground literal l is free if we have $l \vee not\ l \leftarrow$.

A program P is *consistent* if it has an answer set. For a unary predicate p, appearing in P, p is *satisfiable* w.r.t. P if there exists an answer set $(\mathcal{H}, M)$ of P such that $p(a) \in M$ for some $a \in \mathcal{H}$. For a ground literal α, we have $P \models \alpha$ if for all answer sets $(\mathcal{H}, M)$ of P, $\alpha \in M$. Checking whether $P \models \alpha$ is called *query answering*. We can reduce query answering to consistency checking, i.e. $P \models \alpha$ iff $P \cup \{not\ \alpha \leftarrow\ \}$ is not consistent. Consistency checking can be reduced to satisfiability checking, by introducing some new free predicate p.

Finally, note that satisfiability checking for DLPs under the open answer set semantics is undecidable since the undecidable *domino problem*[4] can be reduced to it[17].

3 Adding Grounded Rules to Conceptual Logic Programs

In [17], the syntax of DLPs was restricted in order to regain decidability of reasoning and to enable a reduction of reasoning to normal answer set programming, resulting in *conceptual logic programs (CLPs)*. We recall the intuition and definition of CLPs.

Consider a program P_1 defining when one cheats one's spouse, i.e. if one is married to someone that is different than the person one is dating. We have a specialized rule saying when one is cheating one's spouse with the spouse's friend Jane. Furthermore, some justice is introduced by a constraint ensuring that cheaters will in turn be cheated.

[3] We omit the subscript if it is clear from the context and, furthermore, we will usually write T instead of $T(\emptyset)$.

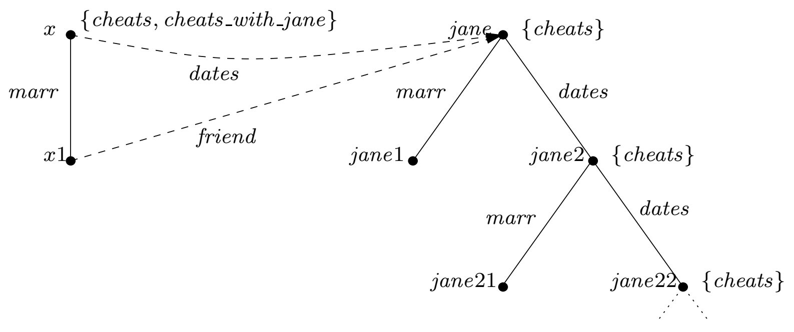

Fig. 1. Forest-Model

$$
\begin{aligned}
cheats(X) &\leftarrow marr(X, Y_1), dates(X, Y_2), Y_1 \neq Y_2 \\
cheats\_with\_jane(X) &\leftarrow marr(X, Y), friend(Y, jane), dates(X, jane), Y \neq jane \\
&\leftarrow cheats(X), dates(X, Y), not\ marr(X, Y), not\ cheats(Y)
\end{aligned}
$$

with $marr$, $friend$ and $dates$ free predicates. An (infinite) answer set of this program that satisfies $cheats\_with\_jane$ is depicted in Figure 1, where e.g. $cheats$ in the label of x indicates that $cheats(x)$ is in the answer set. One sees that x cheats his spouse with Jane since x dates Jane but is married to $x1$. Furthermore, by the constraint, we must have that Jane is also a cheater, and thus, by minimality of answer sets, we must have that Jane is married to some $jane1$ and dates $jane2$, who in turn must be cheating, resulting in an infinite answer set[4]. Formally, a CLP is a DLP consisting of the following types of rules[17]:

- *free rules* $l \vee not\ l \leftarrow$ for a literal l,
- *unary rules* $a(s) \leftarrow \beta(s), \cup_m \gamma_m(s, t_m), \cup_m \delta_m(t_m), \cup_{i \neq j} t_i \neq t_j$, 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* $\leftarrow a(s)$.

where i and j are within the range of m. Note that the example program P_1 is not directly a CLP due to the presence of the literals $marr(X, Y), friend(Y, jane)$ in the second rule where $jane$ is not directly connected to X, as is required for unary rules. However, we can easily rewrite it as a CLP rule by replacing $friend(Y, jane)$ by some $a(Y)$ and adding the unary rule $a(Y) \leftarrow friend(Y, jane)$. In general, programs where the rules have a tree-like body can be easily rewritten as CLPs. Although CLPs allow only constraints of the very simple form $\leftarrow a(s)$ we can easily reduce more complicated constraints $\leftarrow \beta$ to a CLP rule by introducing the unary rule $a(s) \leftarrow \beta$ and $\leftarrow a(s)$.

CLPs were designed to ensure the *forest-model property* (and to a lesser extent the bounded finite model property, cfr. infra). This forest-model property ensures that if a CLP has an answer set where a certain unary predicate is satisfied, then there must be an

[4] We represent the n successors of a node x, as $x1, \ldots, xn$.

answer set that has the form of a forest such that the predicate is true at the root of a tree in such a forest. E.g., the answer set in Figure 1 consists of a tree with an *anonymous*[5] element x as root and the constant *jane* as the root of another tree. It appears that the clean forest structure (i.e. disjoint trees) is perturbed by the connections between x, $x1$ and $jane$. However, it is easy to see that we can encode e.g. $dates(x, jane)$ as $dates^a(x)$ and thus keep $dates^a$ in the label of x. Since there are only a finite number of constants in a program, the labels of the trees are also finite. In effect, a forest-model is a set of trees, with arbitrary connections from elements to constants. As a consequence, the connections between constants, i.e. the roots of the trees, may form an arbitrary graph.

A particular forest-model constructed from an answer set of a program with n constants contains $n+1$ trees, i.e. one for each constant (which is the root of that tree) and an additional one for some anonymous element that contains the predicate of which satisfiability is being checked.

The rules in a CLP make sure that the forest-model property is valid for CLPs[17]. E.g. one cannot have $p(X) \leftarrow not\, f(X, Y)$, since an answer set $(\{x, y\}, \{p(x)\})$ cannot be transformed into a tree: we have nothing to connect x with y. Similarly, we cannot have $f(X, Y) \leftarrow p(X)$ since, for $p(x)$, this would introduce arbitrary connections between x and all other domain elements y, and thus would clearly violate the tree structure. However, it is allowed to have $p(X) \leftarrow q(a)$ for a constant a, since, intuitively, a is a root of its own tree.

As the tree-like rules impose a rather strict format upon the representation of knowledge, we now extend CLPs by allowing for arbitrary ground DLP rules.

Definition 1. *An* ***extended conceptual logic program (ECLP)*** *P is a program $Q \cup R$, where Q is a CLP and R is a finite ground DLP. We denote Q with $clp(P)$ and R with $e(P)$.*

For example, in addition to P_1, we may have a rule representing that if Leo is married to Jane, Jane dates Felix, and Leo himself is not cheating, then Leo dislikes Felix: $dislikes(leo, felix) \leftarrow marr(leo, jane), dates(jane, felix), not\ cheats(leo)$. This ground rule does not have a tree structure, it relates the three constants in an arbitrary graph-like manner. Note that the ground rules can be full-fledged DLP, i.e. with negation as failure. Moreover, predicates in $e(P)$ may be defined in the CLP $clp(P)$, as is the case for $marr$, $dates$ and $cheats$. Vice versa, we may have predicates appearing in the CLP part that are defined in the ground rule part, e.g. $dislikes$ could appear in the CLP part as a $dislikes(X, Y)$ literal.

ECLPs still have the forest-model property, since, intuitively, graph-like connections between constants are allowed in a forest, which is all the ground part $e(P)$ of an ECLP P can ever introduce.

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

A forest-model of the example ECLP would be the forest-model of Figure 1 with additionally $\{dislikes(leo, felix), marr(leo, jane), dates(jane, felix)\}$. As for CLPs in

[5] I.e. a domain element not appearing as a constant in the program.

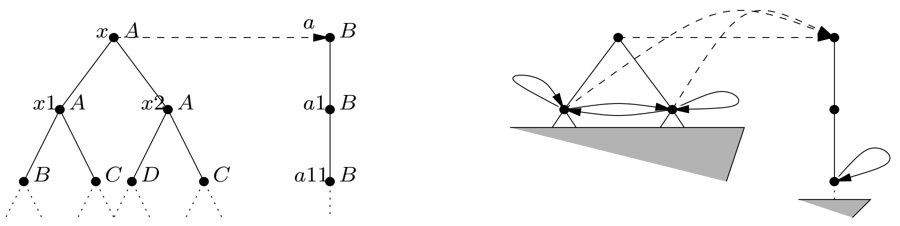

Fig. 2. Cutting a Forest-Model

[17], we would like to establish a bounded finite model property for ECLPs. This property enables the transformation of an (infinite) answer set into a finite one, and, more specifically, it establishes a bound on the number of domain elements that are needed for such a construction. Moreover, this bound depends solely on the program at hand, such that, by introducing a sufficient number of domain elements, we can simulate reasoning with ECLPs by normal finite answer set programming.

We sketch the *cutting* technique from [17] to transform an infinite forest-model into a finite answer set. For every path in a tree in such a forest-model, and every first pair of nodes with equal labels on such a path from the root, we cut away the tree below the second node in the pair and duplicate the outgoing edges of the first node in the second node in the pair. Intuitively, once we encounter on a path a label (a "state") we already encountered, we act as if in the first occurrence of the label instead of going down the tree thereby ignoring the infinite part. For example, Figure 2 shows the cutting of the forest-model on the left, resulting in the finite answer set on the right. Since $x1$ and $x2$ have the same label A as x we replace all outgoing edges from $x1$ and $x2$ with the outgoing edges from x: we have connections from x to $x1$, from x to $x2$, and from x to the constant a. Thus we introduce for xi, $i \in \{1, 2\}$ connections from xi to $x1$, from xi to $x2$, and from xi to a. The tree with constant root is cut in a similar way, but note that one only starts considering duplicate pairs from below the root and thus $(a1, a11)$ is the first pair with duplicate labels to consider. This because it might be that a rule $t(a) \leftarrow$ introduces t in the label of a, however, such a rule cannot be used to motivate the presence of t lower in the tree. Below the root, we would not have this problem as t there would be motivated by a rule with head $t(X)$, which can be matched against any lower node.

Taking into account that forest-models have a finite bounded branching, and that on every path we must always encounter duplicate labels after a bounded depth, together with the fact that there are $n + 1$ trees, for n constants, leads to a finite bound k of needed domain elements, which can be read from the program: the branching can be determined from the branching of the unary rules, and the number of possible labels depends on the number of unary predicates in the program. The number of different labels is exponential in the size of the program such that, taking into account the branching of the program, k is in general double exponential.

However, one has to be cautious with this cutting, e.g. the program with rules $a(X) \leftarrow f(X, Y), a(Y)$, and $a(X) \leftarrow b(X)$ with b and f free, has a tree-model[6]

[6] A tree-model is a forest-model containing only one tree.

$\{a(x), f(x, x1), a(x1), f(x1, x11), a(x11), b(x11)\}$. If one cuts at the first occurrence of a duplicate label, which would be at $x1$ in this case, then $a(x)$ would no longer have a valid support - $b(x11)$ has been cut away - and thus the resulting model would not be minimal. Note that cutting is somewhat similar in spirit to blocking in description logics[3], however, the minimality of answer sets demands some extra precautions, as indicated above.

This problem was solved in [17] for CLPs by enforcing the local model property: forest-models of a CLP should be *locally supported*, i.e. for every literal $q(x)$ $(f(x, y))$ the forest-model can only be motivated by x, one of x's successors, and/or constants. This way, when we cut the trees we never remove the support of any higher nodes in the tree. An extra condition for local supportedness was that a $g(xi, a)$, although it involves only a successor of x and a constant, cannot be in the support of $q(x)$ $(f(x, y))$ since upon cutting at xi, $g(xi, a)$ could be removed while it provides support for $q(x)$ $(f(x, y))$. In the cheating example we have that the forest-model depicted in Figure 1 is not locally supported since $friend(x1, jane)$ is in the support of $cheats\_with\_jane(x)$ - to derive $cheats\_with\_jane(x)$ we need $friend(x1, jane)$.

In the ECLP case, however, where we have an arbitrary ground part, the local model property of [17] is not sufficient. Take, for example, a rule $doesnt\_care(felix) \leftarrow marr(leo, jane), dates(jane, felix), cheats(leo)$, where Felix does not care about dating the married Jane if her husband Leo is cheating as well. Together with the $cheats$ rule from the cheating example, one has that $doesnt\_care(felix)$ is in an answer set if $marr(leo, jane)$, $dates(jane, felix)$,$cheats(leo)$, $marr(leo, leo1)$, and $dates(leo, leo2)$ for successors $leo1$ and $leo2$ of leo are in the answer set. Thus, although the cheats rule in itself does not violate the local model property, adding a ground rule does so, since supports may involve also successors of constants which is not allowed according to the local model property definition for CLPs in [17].

However, cutting of forest-models never removes any successors of constants and, moreover, a successor of a constant is never considered as a candidate for the second node in a duplicate pair since, by definition, the root in a constant tree is not taken into account as a candidate for the first node in a duplicate pair. Thus, we can safely relax the local model property definition from [17] for ECLPs by also allowing successors of constants in the support. In the definition below, we use $\mathcal{H}_{S(l)}$ to denote the domain elements in $S(l)$, the support of l.

Definition 2. *A forest-model* $(\mathcal{H}, M)$ *of an ECLP* P *is* ***locally supported*** *if*
$\forall l = q(x) \in M \vee l = f(x, y) \in M \cdot$
$(\mathcal{H}_{S(l)} \subseteq \{x, xi \mid xi \text{ successor of } x\} \cup \{a, ai \mid a \in \mathcal{H}_P, ai \text{ successor of } a\}) \wedge$
$(\forall f(z, a) \in S(l), a \in \mathcal{H}_P \cdot z \neq xi)$, $p \in upreds(P)$ *is* ***locally satisfiable*** *w.r.t.* P *if there is a locally supported forest-model, a* ***local model*** *for short,* $(\mathcal{H}, M)$ *such that* $p(\varepsilon) \in M$ *for a root* ε *in* $\mathcal{H}$. *An ECLP* P *has the* ***local model property*** *if the following holds: if* $p \in upreds(P)$ *is satisfiable w.r.t.* P *then it is locally satisfiable.*

Thus, a forest-model is locally supported if the support for every $q(x)$ or $f(x, y)$ involves only x itself, successors of x, constants and/or successors of constants. ECLPs with the local model property then have the desired bounded finite model property, i.e. if a (unary) predicate p is satisfiable w.r.t. an ECLP P then it is satisfiable by a finite answer set $(\mathcal{H}, M)$ with $|\mathcal{H}| < k$ where k is solely determined by the program P.

Theorem 2. *Let P be an ECLP with the local model property. Then, P has the bounded finite model property.*

Thanks to this property we can reduce reasoning with ECLPs to normal answer set programming by introducing a sufficiently large bound.

Theorem 3. *Let P be an ECLP with the local model property. $p \in upreds(P)$ is satisfiable w.r.t. P iff there is an answer set M of $\psi(P)$ containing a $p(x_i)$, $1 \leq i \leq k$, where k is as derived above and $\psi(P) = P \cup \{cte(x_i) \leftarrow \ \mid 1 \leq i \leq k\}$.*

The local model property characterizes those ECLPs for which reasoning can be reduced to normal finite answer set programming. However, it is a semantical characterization, which makes it non-trivial to recognize ECLPs satisfying this property. We now identify a class of ECLPs, based on their syntactic structure, that have the local model property.

Local CLPs are CLPs where each unary $a(s) \leftarrow \alpha(s), \gamma_m(s, t_m), \beta_m(t_m), t_i \neq t_j$ and each binary $f(s,t) \leftarrow \alpha(s), \gamma(s,t), \beta(t)$ is such that every $b \in \beta^+_{(m)}$ is either a free predicate, or if $t_{(m)}$ is a constant, $b(t_{(m)})$ is a free literal, or for every $r : b(u) \leftarrow \text{body}(r)$, $\text{body}(r)^+ = \emptyset$. Intuitively, to prove an $a(s)$ ($f(s,t)$) one needs to descend at most one level in the tree, where one can locally prove $a(s)$ ($f(s,t)$), i.e. without the need to go further down the tree. Of course, in the level below s one may need to check more literals which could amount going further down the tree, but whilst doing this one does not need to remember which literals need to be proven above in the tree - in a way a local CLP is memoryless. In [17] local CLPs were shown to have the local model property.

We then define *local ECLPs* as the union of a local CLP and an arbitrary ground DLP.

Definition 3. *A **local ECLP** P is an ECLP where $clp(P)$ is local.*

By the extension of the local model property of CLPs to accommodate for ECLPs, where also successors of constants are allowed in the local support, local ECLPs have the local model property, i.e. the arbitrary ground rules have no influence on the locality.

Theorem 4. *Local ECLPs have the local model property.*

Furthermore, adding a finite number of ground rules to a CLP does not augment the complexity of reasoning.

Theorem 5. *Let P be an ECLP with the local model property. Satisfiability checking w.r.t. P is in* 3-NEXPTIME.

Indeed, we have that the bound k of needed domain elements to simulate reasoning w.r.t. an ECLP P with finite answer set programming is double exponential in the size of P, and thus the size of the translated program $\psi(P)$ (as in Theorem 3) is double exponential in the size of P. Since satisfiability checking w.r.t. $\psi(P)$ is in NEXPTIME w.r.t. the size of the program[9, 5], we have a 3-NEXPTIME bound w.r.t. the size of the original ECLP.

4 Nonmonotonic Ontological and Rule-Based Reasoning with Extended Conceptual Logic Programs

We consider the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ which is the basic DL $\mathcal{ALC}$ with support for role hierarchies ($\mathcal{H}$), nominals/individuals ($\mathcal{O}$), qualified number restrictions ($\mathcal{Q}$), and conjunction ($\sqcap$) and disjunction ($\sqcup$) of roles. $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ is a DL related to the ontology language OWL DL[7], extending it in certain aspects and restricting it in others: OWL DL is a notational variant of the DL $\mathcal{SHOIN}(\mathbf{D})$[18], which adds transitive roles (turning $\mathcal{ALC}$ into $\mathcal{S}$), inverse roles ($\mathcal{I}$), and data types ($\mathbf{D}$) to $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ while removing support for role constructors and qualified number restrictions from it, and allowing only unqualified number restrictions ($\mathcal{N}$).

Formally, the syntax of $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ concept and role expressions can be defined as in Table 1 for concept expressions D, E, concept names A, role expressions R, S, role names Q, and nominals o. The semantics is given by a tuple $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set, representing the set of available domain elements, and $\cdot^{\mathcal{I}}$ is an interpretation function such that $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and $Q^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for concept names A and role names Q, and every nominal o is mapped to some $o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. For complex concept expressions, $\cdot^{\mathcal{I}}$ is defined as in Table 1, where we additionally assume the

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

concept names	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role names	$Q^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
individuals	$\{o\}^{\mathcal{I}} = \{o^{\mathcal{I}}\}$
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	$(\leq n\ R.D)^{\mathcal{I}} = \{x \mid \#\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\} \leq n\}$
	$(\geq n\ R.D)^{\mathcal{I}} = \{x \mid \#\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\} \geq n\}$

unique name assumption for nominals, i.e. if $o_1 \neq o_2$, then $o_1^{\mathcal{I}} \neq o_2^{\mathcal{I}}$. Note that OWL does not have the unique name assumption[26], and thus different individuals can point to the same resource. However, the open answer set semantics gives an Herbrand interpretation to constants, i.e. constants are interpreted as themselves, and for consistency we assume that also DL nominals are interpreted this way. Thus, from a Semantic Web point of view, we assume all individuals are URI's that point to a unique resource.

A DL *knowledge base* consists of *terminological axioms* $C_1 \sqsubseteq C_2$ and *role axioms* $R_1 \sqsubseteq R_2$ for concept expressions C_1 and C_2, and role expressions R_1 and R_2. Axioms express a subset relation: an interpretation $\mathcal{I}$ *satisfies* an axiom $C_1 \sqsubseteq C_2$ ($R_1 \sqsubseteq R_2$) if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ ($R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$). An interpretation is a *model* of a knowledge base Σ if it satisfies every axiom in Σ. A concept C is *satisfiable* w.r.t. Σ if there is a model $\mathcal{I}$ of Σ such that $C^{\mathcal{I}} \neq \emptyset$.

The ontology layer for the Semantic Web is becoming a reality with languages such as OWL DL. Consequently, the rule layer, which provides additional inferencing capabilities on top of DL reasoning, is gaining interest in the Semantic Web community. For example, in [23], integrated reasoning of DLs with *DL-safe* rules was introduced. DL-safe rules are unrestricted Horn clauses where only the communication between the DL knowledge base and the rules is restricted; they enable one to express knowledge inexpressible with DLs alone, e.g. triangular knowledge such as[23]

$$\mathit{BadChild}(X) \leftarrow \mathit{Grandchild}(X), \mathit{parent}(X, Y), \mathit{parent}(Z, Y), \mathit{hates}(X, Z)$$

saying that a grandchild that hates its sibling is a bad child.

We introduce DL-safe rules as in [23]. For a DL knowledge base Σ let N_C and N_R be the concept and role names in Σ and N_P is a set of predicate symbols such that $N_C \cup N_R \subseteq N_P$. A ***DL-atom*** is an atom of the form $A(s)$ or $R(s,t)$ for $A \in N_C$ and $R \in N_R$. A ***DL-safe rule*** is a rule of the form $a \leftarrow b_1, \ldots, b_n$ where a, b_i are atoms and every variable in the rule appears in a non-DL-atom in the rule body. A ***DL-safe program*** is a finite set of DL-safe rules. Let $cts(\Sigma, P)$ be the set of nominals in Σ and constants in P.

The semantics of the combined (Σ, P) for a knowledge base Σ and a DL-safe program P is given by interpreting Σ as a first-order theory $\pi(\Sigma)$, see e.g. [8], every DL-safe rule $a \leftarrow b_1, \ldots, b_n$ as the clause $a \vee \neg b_1 \vee \ldots \vee \neg b_n$, and then considering the first-order interpretation of $\pi(\Sigma) \cup P$. The main reasoning procedure in [23] is *query answering*, i.e. checking whether a ground atom α is true in every first-order model of $\pi(\Sigma) \cup P$, denoted as $(\Sigma, P) \models \alpha$.

We provide an alternative semantics based on DL interpretations as in [19] rather than on first-order interpretations. However, both semantics are compatible as indicated in [23]. For (Σ, P) and an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of Σ we extend $\cdot^{\mathcal{I}}$ for N_P and $\mathcal{H}_P$ such that for unary predicates $p \in N_P$, $p^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, for binary predicates $f \in N_P$, $f^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and $o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for $o \in \mathcal{H}_P$; such an extended interpretation is, by definition, an interpretation of (Σ, P). Furthermore, we impose the unique name assumption such that if $o_1 \neq o_2$, then $o_1^{\mathcal{I}} \neq o_2^{\mathcal{I}}$, for elements $o \in cts(\Sigma, P)$. A *binding* for an interpretation $\mathcal{I}$ of (Σ, P) is a function $\sigma : vars(P) \cup cts(\Sigma, P) \rightarrow \Delta^{\mathcal{I}}$ with $\sigma(o) = o^{\mathcal{I}}$ for $o \in cts(\Sigma, P)$; it maps constants/nominals and variables to domain elements. A unary atom $a(s)$ is then true w.r.t. σ and $\mathcal{I}$ if $\sigma(s) \in a^{\mathcal{I}}$, and a binary atom $f(s,t)$ is true w.r.t. σ and $\mathcal{I}$ if $(\sigma(s), \sigma(t)) \in f^{\mathcal{I}}$. A rule r is satisfied by $\mathcal{I}$ iff for every binding σ w.r.t. $\mathcal{I}$ that makes the atoms in the body true, the head is also true. An interpretation of (Σ, P) is a model if it is a model of Σ and it satisfies every rule in P. Query answering $(\Sigma, P) \models \alpha$ amounts then to checking whether for every (DL) model $\mathcal{I}$ of (Σ, P), the ground atom α is true in $\mathcal{I}$.

In [17], $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ satisfiability checking is reduced to CLP satisfiability checking. Here we reduce query answering w.r.t. $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ extended with DL-safe rules to query answering w.r.t. ECLPs. We first provide some intuition with an example. Take a knowledge base $\Sigma = \{\exists \mathit{manuf\_in}.\mathit{Co} \sqcap \exists \mathit{has\_price} \sqsubseteq \mathit{Product}\}$, expressing that if something is manufactured in some country and it has a price then it is a product. We have some facts in a DL-safe program P about the world we are considering:

$$\begin{array}{rr} \mathit{is\_product\_of}(p, c_1) \leftarrow & \mathit{manuf\_in}(p, \mathit{japan}) \leftarrow \\ \mathit{is\_product\_of}(p, c_2) \leftarrow & \mathit{Co}(\mathit{japan}) \leftarrow \end{array}$$

saying that p is a product of company c_1 and company c_2, that p is manufactured in Japan and that Japan is a country. Those facts are vacuously DL-safe since they do not contain variables. Additionally, we have a DL-safe rule in P saying that if a product is a product of 2 companies then those companies are competitors[7], r_1 : $\mathit{competitors}(C_1, C_2) \leftarrow \mathit{Product}(P), \mathit{is\_product\_of}(P, C_1), \mathit{is\_product\_of}(P, C_2)$. Note that this is indeed a DL-safe rule since every variable occurs in a $\mathit{is\_product\_of}$ atom, which is a non-DL-atom in the body of the rule. The only DL-atom in the rule is $\mathit{Product}(P)$. A possible model $\mathcal{I}$ of (Σ, P) would be $\mathcal{I} = (\{\mathit{japan}, c_1, c_2, p, x\}, \cdot^{\mathcal{I}})$[8] with $\cdot^{\mathcal{I}}$: $\mathit{Co}^{\mathcal{I}} = \{\mathit{japan}\}$, $\mathit{Product}^{\mathcal{I}} = \{p\}$, $\mathit{manuf\_in}^{\mathcal{I}} = \{(p, \mathit{japan})\}$, $\mathit{has\_price}^{\mathcal{I}} = \{(p, x)\}$, $\mathit{is\_product\_of}^{\mathcal{I}} = \{(p, c_1), (p, c_2)\}$ and $\mathit{competitors}^{\mathcal{I}} = \{(c_1, c_2)\}$.

We translate (Σ, P) now to an ECLP: the DL axiom is translated to the constraint $\leftarrow (\exists \mathit{manuf\_in}.\mathit{Co} \sqcap \exists \mathit{has\_price})(X), \mathit{not}\ \mathit{Product}(X)$, where we introduce predicates corresponding to the concept expressions. Furthermore, we define these predicates by the rules

$$\begin{array}{r} (\exists \mathit{manuf\_in}.\mathit{Co} \sqcap \exists \mathit{has\_price})(X) \leftarrow (\exists \mathit{manuf\_in}.\mathit{Co})(X), (\exists \mathit{has\_price})(X) \\ (\exists \mathit{manuf\_in}.\mathit{Co})(X) \leftarrow \mathit{manuf\_in}(X, Y), \mathit{Co}(Y) \\ (\exists \mathit{has\_price})(X) \leftarrow \mathit{has\_price}(X, Y) \end{array}$$

such that if an answer set contains $(\exists \mathit{manuf\_in}.\mathit{Co} \sqcap \exists \mathit{has\_price})(x)$, then, by minimality of answer sets and the first rule, $(\exists \mathit{manuf\_in}.\mathit{Co})(x)$ and $(\exists \mathit{has\_price})(x)$ are in the answer set, and, by the second and third rule, there must be a y_1 and a y_2 such that $\mathit{manuf\_in}(x, y_1)$, $\mathit{Co}(y_1)$, and $\mathit{has\_price}(x, y_2)$ are in the answer set. The opposite direction is also valid, i.e. if $\mathit{manuf\_in}(x, y_1)$, $\mathit{Co}(y_1)$, and $\mathit{has\_price}(x, y_2)$ are in the answer set then $(\exists \mathit{manuf\_in}.\mathit{Co} \sqcap \exists \mathit{has\_price})(x)$ is in the answer set since rules need to be satisfied. This kind of behavior exactly mimics the DL semantics of the corresponding constructs. Furthermore, we introduce the concept and role names by means of free rules, indicating that a domain element (or a pair of domain elements) is of a certain type or not.

$$\begin{array}{r} \mathit{Product}(X) \vee \mathit{not}\ \mathit{Product}(X) \leftarrow \\ \mathit{Co}(X) \vee \mathit{not}\ \mathit{Co}(X) \leftarrow \\ \mathit{manuf\_in}(X, Y) \vee \mathit{not}\ \mathit{manuf\_in}(X, Y) \leftarrow \\ \mathit{has\_price}(X, Y) \vee \mathit{not}\ \mathit{has\_price}(X, Y) \leftarrow \end{array}$$

This concludes the CLP part of the translation of (Σ, P). The ground DLP part consists of the same facts as in the DL-safe part; it also contains the grounding of the rule r_1 in P with constants $\{\mathit{japan}, p, c_1, c_2\}$, e.g. the rule

$$r_2 : \mathit{competitors}(c_1, c_2) \leftarrow \mathit{Product}(p), \mathit{is\_product\_of}(p, c_1), \mathit{is\_product\_of}(p, c_2)$$

[7] Actually, to correspond entirely with the desired semantics, we would need to indicate that C_1 and C_2 are different companies. This seems to be not possible with the DL-safe rules in [23], however, it is with ECLPs using $\neq$.

[8] We take $o^{\mathcal{I}} = o$, $o \in \mathit{cts}(\Sigma, P)$, for ease of notation.

Since DL-safe rules have a first-order interpretation one may have that $(c_1, c_2) \in \mathit{competitors}^{\mathcal{I}}$ for a model $\mathcal{I}$ of (Σ, P) without any justification in $\mathcal{I}$, i.e. the body of r_1 in P does not need to be satisfied in order to have $(c_1, c_2) \in \mathit{competitors}^{\mathcal{I}}$. The answer set semantics however only deduces $\mathit{competitors}(c_1, c_2)$ in an answer set if e.g. the body of r_2 is satisfied in that answer set, since otherwise the answer set would not be minimal (one could omit $\mathit{competitors}(c_1, c_2)$ and still have an answer set).

To solve this, we introduce for each head a of a rule in the ground DLP part, a free rule $a \lor \mathit{not}\ a \leftarrow$, e.g. $\mathit{competitor}(c_1, c_2) \lor \mathit{not}\ \mathit{competitor}(c_1, c_2) \leftarrow$ such that one has always a motivation for $\mathit{competitor}(c_1, c_2)$, mimicking the first-order semantics.

We refer to [17] for the definition of the closure $\mathit{clos}(\Sigma)$ of a $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base Σ, but basically, for a concept expression D in Σ it includes the subconcepts of D. Formally, we define the CLP $\Phi_1(\Sigma, P)$ for a $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base Σ and a DL-safe program P as the program containing for every concept expression $D \in \mathit{clos}(\Sigma)$ the rules in Table 2. Furthermore, for every concept

Table 2. CLP Translation $\Phi_1(\Sigma, P)$

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

name A and role name Q in Σ, we add the free rules $A(X) \lor \mathit{not}\ A(X) \leftarrow$ and $R(X, Y) \lor \mathit{not}\ R(X, Y) \leftarrow$. Nominals o in Σ are handled by introducing predicates $\{o\}$ with facts $\{o\}(o) \leftarrow$ in $\Phi_1(\Sigma, P)$, such that we can only have that $\{o\}(x)$ is in an answer set if $x = o$. $\Phi_1(\Sigma, P)$ is not a local ECLP, but due to the fact that the body of a rule becomes structurally smaller one can transform it to a local ECLP while preserving satisfiability[17].

We define $\Phi_2(\Sigma, P)$ as the ground DLP $P_{cts(\Sigma,P)}$, i.e. P grounded with all constants and nominals in Σ and P, together with free rules $\mathrm{head}(r) \lor \mathit{not}\ \mathrm{head}(r) \leftarrow$ for each $r \in P_{cts(\Sigma,P)}$.

Theorem 6. *For an $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base Σ and a DL-safe program P, we have $(\Sigma, P) \models \alpha$ iff $\Phi_1(\Sigma, P) \cup \Phi_2(\Sigma, P) \models \alpha$.*

In [23] the $\mathcal{SHOIN}(\mathbf{D})$ DL is considered instead of $\mathcal{ALCHOQ}(\sqcup, \sqcap)$, which extends and at the same time restricts the type of allowed constructors. DL-safe rules allow for variables, however, this does not make them more expressive than ground DLP programs: [23] proves that $(\Sigma, P) \models \alpha$ iff $(\Sigma, P^g) \models \alpha$ where P^g is the grounding of P w.r.t. constants and nominals in (Σ, P). Moreover, using ECLPs instead of a DL knowledge base with DL-safe rules on top has the further advantage of nonmonotonicity by means of negation as failure in both the CLP part and the grounded DLP part, whereas both DLs and DL-safe rules are monotonic (DL-safe rules are Horn clauses and thus do not allow for negation as failure).

5 Related Work

We highlight some of the current research trends on the application of nonmonotonicity to the Semantic Web and refer the reader for further related work on the combination of (not necessarily nonmonotonic) rules and ontologies to [17].

In [2], one builds a nonmonotonic rule system on top of the ontology language DAML+OIL[6], a predecessor of OWL. More specifically, they use *defeasible logic*[24] to express rule-based knowledge and argue its use for E-commerce applications on the Semantic Web. Another approach combining DAML+OIL with rules can be found in [15], where *situated courteous logic programs* in the rule markup language RuleML[1] provide for the nonmonotonic rule system.

[10] combines the expressive $\mathcal{SHOIN}(\mathbf{D})$, i.e. OWL DL, with ASP reasoning by considering the DL knowledge base as a black box that can be queried from the rules. Moreover, inferences made by rules can serve as input to the DL knowledge base as well, leading to a bidirectional flow of information. A disadvantage of this approach, as was remarked in [23], is that, since one considers only consequences of the DL knowledge base, i.e. atoms that are true in all models, some more fine-grained inferences will not be made by the rules. Since reasoning with CLPs can be reduced to finite ASP, it can be trivially reduced to the approach in [10] with an empty DL knowledge base. In [11] the approach of [10] was adapted for the well-founded semantics instead of the answer set semantics.

[14] explains how reasoning with SWRL[20], i.e. OWL extended with Datalog in RuleML, can be done by iteratively calling the DL reasoner RACER[16] and the rule-based reasoner *Jess*[12], each feeding the other with the inferences it made. Since SWRL is undecidable, and such an iterative procedure is thus incomplete, it shows that intractable worst-case complexity (or even undecidability) should not hold one back to device practical and useful combined reasoners. A similar iterative angle is taken in [22] where SWRL is extended with negation as failure and equipped with an answer set semantics, resulting in a nonmonotonic but undecidable system.

6 Conclusions and Directions for Further Research

We extended CLPs with a finite set of arbitrary ground DLP rules, and showed that reasoning with the resulting ECLPs can be reduced to finite answer set programming. We established an upper complexity bound and simulated reasoning in a DL equipped with DL-safe rules.

The upper 3-NEXPTIME bound for reasoning with ECLPs is rather bad, however, encouraged by practical algorithms for highly intractable DL algorithms, we believe that, using heuristics, one can also implement practical reasoners for ECLPs. This is subject for further research.

References

1. The Rule Markup Initiative. http://www.ruleml.org.
2. G. Antoniou. A Nonmonotonic Rule System using Ontologies. CEUR Proceedings, 2002.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.
4. F. Baader and U. Sattler. Number Restrictions on Complex Roles in Description logics. In *Proc. of KR-96*, pages 328–339, 1996.
5. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.
6. S. Bechhofer, C. Goble, and I. Horrocks. DAML+OIL is not Enough. In *Proc. of the First Semantic Web Working Symposium (SWWS'01)*, pages 151–159. CEUR, 2001.
7. 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. `http://www.w3.org/TR/owl-ref/`, 2004.
8. A. Borgida. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.
9. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
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. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Well-Founded Semantics for Description Logic Programs in the Semantic Web. In *Proc. of RuleML 2004*, number 3323 in LNCS, pages 81–97. Springer, 2004.
12. E.J. Friendman-Hill. Jess homepage. http://herzberg.ca.sandia.gov/jess/.
13. 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.
14. C. Golbreich. Combining Rule and Ontology Reasoners for the Semantic Web. In *Proc. of RuleML 2004*, number 3323 in LNCS, pages 6–22. Springer, 2004.
15. B. N. Grosof and T. C. Poon. SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions. In *Proc. of WWW 2003*, pages 340–349. ACM Press, 2003.
16. V. Haarslev and R. Moller. Description of the RACER System and its Applications. In *Proc. of Description Logics 2001*, 2001.
17. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Semantic Web Reasoning with Conceptual Logic Programs. In *Proc. of RuleML 2004*, number 3323 in LNCS, pages 113–127. Springer, 2004.
18. I. Horrocks and P. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *J. of Web Semantics*, 2004. To Appear.
19. I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proc. of WWW 2004*. ACM, 2004.
20. 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.
21. N. Leone, W. Faber, and G. Pfeifer. DLV homepage. http://www.dbai.tuwien.ac.at/proj/dlv/.
22. J. Mei, S. Liu, A. Yue, and Z. Lin. An Extension to OWL with General Rules. In *Proc. of RuleML 2004*, number 3323 in LNCS, pages 6–22. Springer, 2004.

23. Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. In *Proc. of ISWC 2004*, number 3298 in LNCS, pages 549–563. Springer, 2004.
24. D. Nute. Defeasible Logic. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming (Vol. 3)*, pages 353–395. Clarendon Press, 1994.
25. P. Simons. Smodels homepage. http://www.tcs.hut.fi/Software/smodels/.
26. M. Smith, C. Welty, and D. McGuinness. OWL Web Ontology Language Guide. `http://www.w3.org/TR/owl-guide/`, 2004.