Open Answer Set Programming for the Semantic Web

Stijn Heymans [a] Davy Van Nieuwenborgh [a] Dirk Vermeir [a,*]

[a] *Dept. of Computer Science*
Vrije Universiteit Brussel, VUB
Pleinlaan 2, B1050 Brussels, Belgium

Abstract

We extend answer set programming (ASP) with, possibly infinite, open domains. Since this leads to undecidable reasoning, we restrict the syntax of programs, while carefully guarding knowledge representation mechanisms such as negation as failure and inequalities. Reasoning with the resulting extended forest logic programs (EFoLPs) can be reduced to finite answer set programming, for which reasoners are available.

We argue that extended forest logic programming is a useful tool for uniformly representing and reasoning with both ontological and rule-based knowledge, as they can capture a large fragment of the OWL DL ontology language equipped with DL-safe rules. Furthermore, EFoLPs enable nonmonotonic reasoning, a desirable feature in locally closed subareas of the Semantic Web.

Key words: Answer Set Programming, Semantic Web, Description Logics, Open Domain Reasoning

1 Introduction

Answer set programming (ASP) [21] is a logic programming 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 problematic

[*] Corresponding author: Tel. +32 2 6293755, Fax. +32 2 6293525
Email addresses: sheymans@vub.ac.be (Stijn Heymans), dvnieuwe@vub.ac.be (Davy Van Nieuwenborgh), dvermeir@vub.ac.be (Dirk Vermeir).

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 P:

$$\begin{aligned} bigCar(X) &\leftarrow Manager(X) \\ Manager(X) \lor not\ Manager(X) &\leftarrow \\ \neg Manager(felix) &\leftarrow \end{aligned}$$

Grounding with the only present constant, $felix$, yields the program

$$\begin{aligned} bigCar(felix) &\leftarrow Manager(felix) \\ Manager(felix) \lor not\ Manager(felix) &\leftarrow \\ \neg Manager(felix) &\leftarrow \end{aligned}$$

which has a single answer set $\{\neg Manager(felix)\}$ such that one wrongfully concludes that there are never managers that drive big cars: the conclusions of the program depend on the present instance data.

We resolve this by introducing, possibly infinite, *open domains*. Under the *open answer set semantics* the example has an open answer set

$$(\mathcal{H} = \{felix, heather\}, \\ M = \{\neg Manager(felix), Manager(heather), bigCar(heather)\})\ ,$$

where $\mathcal{H}$ is a *universe* for P that extends the constants present in P and M is an answer set of P grounded with $\mathcal{H}$. One concludes that it is possible that there are persons that are managers and thus drive big cars, corresponding to the intended semantics of the program. The open answer set semantics enables data independent reasoning: an ontology engineer does not need to introduce all significant constants in the program, which allows her to concentrate on modeling the ontological knowledge only. Note the use of disjunction and negation as failure in the head of $Manager(X) \lor not\ Manager(X) \leftarrow$. 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 support for the presence of anonymous individuals, i.e., elements that are not constants in the program, allows to bridge the semantics of logic programming and *description logics* [5]: open answer set programming enables both a nonmonotonic semantics (typical for logic programming paradigms) and the use of *open domains*, one of the key features for conceptual modeling, as present in classical logics.

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) extended forest logic programs (EFoLPs)*, ensures a reduction of reasoning to finite, closed, ASP by virtue of the forest-model property and the bounded finite model property. EFoLPs are thus amenable for reasoning with existing answer set solvers such as DLV [41] and SMODELS [50].

Reasoning with both ontological knowledge, in the form of a description logic (DL) [5] 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., [45] 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 express triangular knowledge that is not expressible with DLs alone: $uncle(a, c) \leftarrow brother(a, b), parent(b, c)$. Note that DL-safe rules can contain variables but, by DL-safeness, the rules correspond to their grounded version where the grounding is done w.r.t. the present constants and nominals in the rules and DL knowledge base. It does not take into account anonymous domain elements, which is a serious limitation. On the other hand allowing for a grounding with anonymous elements would immediately yield undecidability.

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 particular business rules, e.g.,

$$i : invest(tps, 10K) \leftarrow not\ good\_product(tps)$$

saying that if its particular top selling product tps cannot be 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

$$R_1 \equiv \{complaint(tps) \leftarrow\}\ ,$$

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 forest logic programming (EFoLP)* to express the above knowledge. Intuitively, any model of $\mathcal{O} \cup \{i\} \cup R_1$, must verify $complaint(tps)$, and $good\_product(X) \leftarrow not\ complaint(X)$ 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

$$R_2 \equiv \{complaint(tps) \leftarrow , good\_product(tps) \leftarrow \}$$

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 well-defined environment with dynamically changing knowledge.

EFoLPs are defined as pairs (Q, R) consisting of, on the one hand, a *forest logic program (FoLP)* Q capable of expressing conceptual knowledge, as in, e.g., DL knowledge bases, and, on the other hand, a finite arbitrary program R which allows to relate constants/individuals in arbitrary ways. An *EFoLP answer set* of such a (Q, R) is defined as an open answer set of $Q \cup R'$, where R' is the program R grounded with constants from $Q \cup R$. On the semantical level, an EFoLP corresponds to a FoLP with a finite set of ground arbitrary rules. Syntactically, however, the pair notation allows for a more compact representation. Intuitively, an EFoLP consists of a syntactically restricted part allowing open domain reasoning and an arbitrary part where reasoning is on the present constants only. In particular, EFoLPs can simulate reasoning in the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$, a DL closely related to OWL [9], equipped with DL-safe rules, Moreover, EFoLPs are capable, as indicated above, of nonmonotonic reasoning as well, since they allow for negation as failure both in the FoLP part as in the arbitrary rule part.

Note that, although we allow for negation as failure, (E)FoLPs still have to satisfy rather strict syntactical restrictions to ensure the forest-model property. E.g., the above *uncle* relationship cannot be expressed with variables that can be ground by anynomous elements. We do allow for arbitrary rules in EFoLPs, however, their variables must be grounded with constants (either from the program or from the DL knowledge base), which makes their usefulness rather limited. The alternative, i.e., loosening up the syntactical restrictions or allowing grounding with anonymous elements in the arbitrary rules, easily leads to undecidability.

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) EFoLPs, reduce reasoning to normal ASP, and establish complexity results. In Section 4, we show the EFoLP simulation of an expressive class of DLs equipped with DL-safe rules. Section 5 relates other work to our approach. Finally, Section 6 contains conclusions and directions for further research.

2 Open Answer Set Programming

In Subsection 2.1, we introduce the open answer set semantics; Subsection 2.2 argues the undecidability of reasoning under the open answer set semantics, and Subsection 2.3 defines the class of *acyclic* programs, which will be useful for the simulation of DLs in Section 4.

2.1 Basic Definitions and Results

A *term* is either a *constant* or a *variable*, and is denoted by a string of letters where a constant starts with a lower-case letter and a variable with an upper-case letter. An *atom* is of the form $a(t)$ or $f(s,t)$ where a is a unary predicate name, f is a binary predicate name, and s and t are terms. A *literal* is an atom or an atom preceded with the classical negation symbol $\neg$. We assume $\neg\neg a \equiv a$ for an atom a; for a set of literals α, $\neg\alpha \equiv \{\neg l | l \in \alpha\}$.

An *extended literal* is a literal or a literal preceded by the *negation as failure (naf)* symbol *not*. We will often denote a set of unary extended literals, ranging over a common term s, as $\alpha(s)$, e.g., $\{a(s), not\ b(s)\}$ may be denoted as $\{a, not\ b\}(s)$. 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 $\beta^+ \equiv \{l \mid l \in \beta, l \text{ literal}\}$, the negative part is $\beta^- \equiv \{l \mid not\ l \in \beta\}$, e.g., for $\beta = \{a, not\ \neg b, not\ c\}$, we have that $\beta^+ = \{a\}$ and $\beta^- = \{\neg b, c\}$. Furthermore, we assume the existence of a binary predicate $\neq$, with the usual interpretation.

A *disjunctive extended logic program* (DLP) is a countable set of rules $\alpha \leftarrow \beta$ where α and β are finite sets of extended literals and $|\alpha^+| \leq 1$; as usual, α is supposed to be a disjunction of extended literals and β a conjunction. In contrast to the DLP we define here, classical DLP allows for α to be an arbitrary set of extended literals; our extra condition ensures that the GL-reduct, defined below, is disjunction-free, which avoids the use of an extra NP oracle in satisfiability checking, see Section 3.3. If $\alpha = \emptyset$, we call the rule a *constraint*. The set α is the *head* of the rule while β is called the *body*, denoted, for a rule r, as $\text{head}(r)$ and $\text{body}(r)$ respectively. As usual, atoms, (extended) literals, rules, and programs that do not contain variables are *ground*. A set of ground literals X is *consistent* if $X \cap \neg X = \emptyset$.

Note that programs with *not* in the head can be rewritten as equivalent programs without *not* in the head [39]. Since the former programs have a non-minimal semantics, useful for introducing the types in conceptual models, while the latter programs have not, they are more appropriate in the context of conceptual knowledge representation.

For a DLP P, let $\mathcal{H}_P$ be the constants in P. A (possibly infinite) non-empty countable set of constants $\mathcal{H}$ such that $\mathcal{H}_P \subseteq \mathcal{H}$, is called a *universe* for P. We denote $P_\mathcal{H}$ the *ground program* obtained from P by substituting every variable in P by every possible constant in $\mathcal{H}$ such that the inequalities are true (and subsequently removed). For a program P and its constants $\mathcal{H}_P$, we will denote $P_{\mathcal{H}_P}$ often simply as $ground(P)$.

Example 1 *The program P*

$$sel(I,S) \lor not\ sel(I,S) \leftarrow \qquad\qquad av(i) \leftarrow$$
$$av(I) \leftarrow sel(I,S)$$

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 are $\mathcal{H}_P = \{i\}$; some of the universes for P are $\mathcal{H}_1 = \{i, s\}$ or an infinite $\mathcal{H}_2 = \{i, x_1, x_2, \ldots\}$.

Let $\mathcal{L}_P$ be the set of literals that can be formed from a grounded program P, $preds(P)$ are the predicate names in P, and $upreds(P)$ and $bpreds(P)$ the unary and binary predicate names respectively; unless specified otherwise, $\neg p$, for a predicate name p, is also considered to be a predicate name.

An *interpretation* I of a ground 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 ground program P, I is a *model* of P if I satisfies every rule in P. We define the *GL-reduct* [42] w.r.t. I 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 ground P if I is the subset minimal model of P^I. An *open interpretation* of P is a pair $(\mathcal{H}, M)$ where $\mathcal{H}$ is a universe for P and M 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 $P_\mathcal{H}$. We denote this as $(\mathcal{H}, M) \models P$.

Example 2 *Considering the program P from Example 1, we have that, with a universe $\mathcal{H} = \{i, s, x\}$ for P, $(\mathcal{H}, M_1 = \{av(i), sel(x,s), av(x)\})$ and $(\mathcal{H}, M_2 = \{av(i)\})$ are some open answer sets of P. Since M_1 contains a literal $sel(x,s)$, the GL-reduct $P_\mathcal{H}^{M_1}$ contains $sel(x,s) \leftarrow$, which motivates the presence of $sel(x,s)$ in M_1. On the other hand, since $sel(x,s) \notin M_2$, $sel(x,s) \lor not\ sel(x,s) \leftarrow$ is* *satisfied and is not 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 $\leftarrow sel(x, s)$ would make sure that no answer set contains $sel(x, s)$. We 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 $l \vee not\ l \leftarrow$.*

In the following, we usually omit the "open" qualifier and assume that programs are finite unless they are the result of grounding with an infinite universe. 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 $a \in \mathcal{H}$. Consistency checking can be reduced to satisfiability checking by introducing some new predicate: for a program P and a program $P' = P \cup \{p(X) \vee not\ p(X) \leftarrow\}$ with p not appearing in P, we have that P is consistent iff p is satisfiable w.r.t. P'. 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.

There are programs such that a predicate is only satisfiable w.r.t. that program by an infinite open answer set.

Example 3 *The program*

$$\begin{array}{lrl}
r_1: & restore(X) & \leftarrow crash(X), y(X, Y), backSucc(Y) \\
r_2: & backSucc(X) & \leftarrow \neg crash(X), y(X, Y), not\ backFail(Y) \\
r_3: & backFail(X) & \leftarrow not\ backSucc(X) \\
r_4: & & \leftarrow y(Y_1, X), y(Y_2, X), Y_1 \neq Y_2 \\
r_5: & y(X, Y) \vee not\ y(X, Y) & \leftarrow \\
r_6: & crash(X) \vee not\ crash(X) & \leftarrow \\
r_7: & \neg crash(X) \vee not\ \neg crash(X) & \leftarrow
\end{array}$$

represents the knowledge that a system that has crashed on a particular day, can be restored on that day if a backup of the system on the day before succeeded. Backups succeed, if the system does not crash and it cannot be established that the backups at previous dates failed. Rules r_1, r_2, and r_3 express the above knowledge, and r_4 ensures that for a particular today there can be only one tomorrow (y stands for yesterday*). Every open answer set $(\mathcal{H}, M)$ of this program that makes $restore$ satisfiable, i.e., such that there is a $restore(x) \in M$ for $x \in \mathcal{H}$, must be infinite. An* *example of such an answer set M is (we omit $\mathcal{H}$ if it is clear from M)*

$$\{restore(x), crash(x), backFail(x), y(x, x_1), \\ backSucc(x_1), \neg crash(x_1), y(x_1, x_2) \\ backSucc(x_2), \neg crash(x_2), y(x_2, x_3), \ldots\}$$

One sees that every $backSucc$ literal with element x_i enforces a new y-successor x_{i+1} since none of the previously introduced universe elements can be used without violating rule r_4.

Although we allow for infinite universes, we can finitely motivate the presence of literals in answer sets. We express the motivation of a literal more formally by an *immediate consequence 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, $T_{P^M_{\mathcal{H}}} : \mathcal{L}_{P^M_{\mathcal{H}}} \rightarrow \mathcal{L}_{P^M_{\mathcal{H}}}$ is defined as $T(B) \equiv B \cup \{a | a \leftarrow \beta \in P^M_{\mathcal{H}} \wedge \beta \subseteq B\}$, where we omitted the subscript from $T_{P^M_{\mathcal{H}}}$. Additionally, we have $T^0(B) \equiv B$, and $T^{n+1}(B) \equiv T(T^n(B))$. We usually write T^n instead of $T^n(\emptyset)$.

Theorem 4 *Let P be a DLP and $(\mathcal{H}, M)$ an open answer set of P. Then, $\forall a \in M \cdot \exists n < \infty \cdot a \in T^n$.*

PROOF. Assume $\exists a_1 \in M \cdot \forall n < \infty \cdot a_1 \notin T^n$. One can then construct an infinite sequence $\{a_1, a_2, \ldots\} \subseteq M$ such that $\forall i \cdot \forall n < \infty \cdot a_i \notin T^n$. The constructed answer set $M' \equiv M \backslash \{a_1, a_2, \ldots\}$ is a model of $P^M_{\mathcal{H}}$, contradicting the minimality of M. $\Box$

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 $S^n(a) \equiv \{a\}$ and $S^k(a) \equiv \{\beta \mid b \leftarrow \beta \in P^M_{\mathcal{H}}, \beta \subseteq T^k, \beta \nsubseteq T^{k-1}, b \in S^{k+1}(a)\}$, $1 \leq k < n$. The support for a is $S(a) \equiv \cup^n_{k=1} S^k(a)$.

Example 5 *For Example 3, $\{crash(x), y(x, x_1), \neg crash(x_1), y(x_1, x_2)\} \subseteq T^1$, $backSucc(x_1) \in T^2$, and $restore(x) \in T^3$, such that*

$$\begin{aligned} S(restore(x)) &= S^3(restore(x)) \cup S^2(restore(x)) \cup S^1(restore(x)) \\ &= \{restore(x)\} \cup \{crash(x), y(x, x_1), backSucc(x_1)\} \\ &\cup \{\neg crash(x_1), y(x_1, x_2)\} \ . \end{aligned}$$

indicates which literals were responsible for the presence of $restore(x)$ in the answer set.

Satisfiability checking for DLPs under the open answer set semantics is undecidable since the undecidable *domino problem* [6] can be reduced to it. In the domino problem, one has a finite set of domino types $D = \{D_1, \ldots D_m\}$ and two relations indicating which domino types may be placed side by side horizontally, $H \subseteq D \times D$, and vertically, $V \subseteq D \times D$. The domino problem is the search for a tiling, compatible with H and V, of the plane $\mathbb{N} \times \mathbb{N}$, i.e., a $t : \mathbb{N} \times \mathbb{N} \to D$ s.t. $(t(m,n), t(m+1,n)) \in H$ and $(t(m,n), t(m,n+1)) \in V$ for every $m, n \in \mathbb{N}$.

We omit the detail of the reduction but note the representation of the plane since this already unveils an important source of undecidability. The plane $\mathbb{N} \times \mathbb{N}$ can be represented by predicates h and v, where $h(X,Y)$ and $v(X,Y)$ indicate that Y is $X+1$ for X along the horizontal (resp. vertical) axis. Every tile has only one h-successor, such that we have a $\leftarrow h(X,Y_1), h(X,Y_2), Y_1 \neq Y_2$, and every tile has at least one such successor: $h1(X) \leftarrow h(X,Y)$ and $\leftarrow \mathit{not}\ h1(X)$. The same holds for v. Furthermore, taking one step in the vertical direction followed by a horizontal step should be the same as the opposite action: $\mathit{seq}(X,Z) \leftarrow h(X,Y), v(Y,Z)$; $\mathit{seq}(X,Z) \leftarrow v(X,Y), h(Y,Z)$; $\leftarrow \mathit{seq}(X,Z_1), \mathit{seq}(X,Z_2), Z_1 \neq Z_2$.

Checking for a compatible tiling can then be done by introducing unary predicates for each domino type, checking the compatibility locally at each tile, and making sure that each tile can be reached. The main problem, however, are the 2 seq-rules which express composition of binary predicates; without those, we would have a DLP for which satisfiability checking is decidable.

2.3 Acyclic Programs

For the translation of description logics to open answer set programming in Section 4, we need the additional terminology of *acyclic programs*, i.e., programs that do not allow recursion through positive literals.

Formally, a *dependency graph* DG_P for a DLP P is defined by edges between predicates a and b such that $a \to b$ iff there is a rule $\alpha \leftarrow \beta \in P$ such that a is a predicate from α^+ and b is a predicate from β^+. A DLP P is *positively acyclic*, acyclic for short, if DG_P does not contain cycles. An important distinction with stratified programs [7] is that recursion through negated literals is still allowed.

A useful property of acyclic programs, as we will see in Section 4, is that they can be rewritten such that there appear no positive unary literals in the body anymore; one replaces them by a double negation. Formally, for an acyclic program P, we define $\phi(P)$ as the program P with rules $r : \alpha \leftarrow \beta, \gamma$, for $\alpha \neq \emptyset$ and β the unary literals of body(r), replaced by $\alpha \leftarrow \mathit{not}\ \beta', \gamma$ and $b'(X) \leftarrow \mathit{not}\ b(X)$, for all

$b'(X) \in \beta'$ where $\beta' = \{b'(X) \mid b(X) \in \beta\}$.

Theorem 6 *Let P be an acyclic program and $p \in upreds(P)$. p is satisfiable w.r.t. P iff p is satisfiable w.r.t. $\phi(P)$.*

PROOF. For the "only if" direction, assume p is satisfiable w.r.t. P, i.e., there is an open answer set $(\mathcal{H}, M)$ of P such that $p(a) \in M$. One can show that $(\mathcal{H}, M')$ with $M' = M \cup \{b'(x) \mid b(x) \notin M, b' \in \phi(P)\}$ is an answer set of $\phi(P)$.

For the "if" direction, assume p is satisfiable w.r.t. $\phi(P)$, i.e., there is an open answer set $(\mathcal{H}, M)$ of $\phi(P)$ such that $p(a) \in M$. Define $M' = M \setminus \{b'(x)\}$, then $(\mathcal{H}, M')$ is an answer set of P and $p(a) \in M'$. $\square$

Example 7 *Take the program P*

$$\begin{aligned} a(X) &\leftarrow b(X), f(X,Y), not\ c(Y) \\ b(X) \vee not\ b(X) &\leftarrow \\ f(X,Y) \vee not\ f(X,Y) &\leftarrow \end{aligned}$$

The dependency graph of this program is $\{a \rightarrow b, a \rightarrow f\}$ such that P is acyclic. The translation $\phi(P)$ is then

$$\begin{aligned} a(X) &\leftarrow not\ b'(X), f(X,Y), not\ c(Y) \\ b'(X) &\leftarrow not\ b(X) \\ b(X) \vee not\ b(X) &\leftarrow \\ f(X,Y) \vee not\ f(X,Y) &\leftarrow \end{aligned}$$

which has, among others, the answer set $(\{x,y\}, \{a(x), b(x), f(x,y), b'(y)\})$, corresponding to an answer set $(\{x,y\}, \{a(x), b(x), f(x,y)\})$ of P.

Theorem 6 is in general not valid for programs that are not acyclic.

Example 8 *Consider the program P*

$$a(X) \leftarrow a(X)$$

This is not an acyclic program and $\phi(P)$ is the program

$$\begin{aligned} a(X) &\leftarrow not\ a'(X) \\ a'(X) &\leftarrow not\ a(X) \end{aligned}$$

with an answer set $(\{x\}, \{a(x)\})$*, which does not correspond to any answer set of* P.

3 Extended Forest Logic Programs

In Subsection 3.1, we introduce the *forest-model property* and define a syntactically restricted class of programs, *forest logic programs (FoLPs)* [28], satisfying this property. We show in Subsection 3.2 that a particular type of FoLPs, FoLPs with the local model property, has the *bounded finite model property*, which enables a reduction to finite ASP. Subsection 3.3 identifies an upper bound for the complexity of reasoning. Finally, in Subsection 3.4, we extend FoLPs with an arbitrary finite set of rules that can only be grounded with constants present in the program, resulting in EFoLPs [29], and show that properties such as the forest-model property and the bounded finite model property remain valid.

3.1 Forest-model Property

As seen in the previous section, open answer set programming is rather powerful, even to the extent that satisfiability checking in the general case is undecidable. As in modal logics, the so-called *tree-model property* will prove to be a critical factor in showing decidability of satisfiability checking [53]. Roughly, a program has the tree-model property if one has that if there are answer sets that make a predicate satisfiable there must also be answer sets with a tree-structure that make the predicate satisfiable. A generalization of this property is the *forest-model property*: if there is an answer set that makes a predicate satisfiable, then there is an answer set that has the form of a set of trees, a forest. A similar property arises for DLs that include nominals, e.g., $\mathcal{SHOQ}(\mathbf{D})$[34].

For a $x \in \mathbb{N}_0^*$, i.e., a finite sequence of natural numbers, we denote the concatenation of a number $c \in \mathbb{N}$ to x as $x \cdot c$, or, abbreviated, as xc. Formally, a *(finite) tree* T is a (finite) subset of $\mathbb{N}_0^*$ such that if $x \cdot c \in T$ for $x \in \mathbb{N}_0^*$ and $c \in \mathbb{N}_0$, we have that $x \in T$. Elements of T are called nodes and the empty word ε is the root of T. For a node $x \in T$ we call $x \cdot c \in T$, $c \in \mathbb{N}_0$, *successors* of x. By convention, $x \cdot 0 = x$ and $(x \cdot c) \cdot -1 = x$ ($\varepsilon \cdot -1$ is undefined). If every node x in a tree has k successors we say that the tree is *k-ary*. E.g. $T_1 = \{\varepsilon, \varepsilon 1, \varepsilon 2, \varepsilon 11\}$ is a finite tree with root ε, two successors $\varepsilon 1$ and $\varepsilon 2$, and $\varepsilon 11$ a successor of $\varepsilon 1$; T_1 will also be written as $\{\varepsilon, 1, 2, 11\}$. A *labeled tree* over an alphabet Σ is a tuple (T, t) where T is a tree and $t : T \to \Sigma$ is a labeling function; usually we will identify the tree (T, t) with t and we will write t_x for trees where the root is identified with x: if the root in T_1 is a constant a, we write it as $\{a, a1, a2, a12\}$, and a labeling function for T_1 is denoted as t_a. A *forest* F is a finite multi-set $\{t_{x_1}, \dots, t_{x_n}\}$, with each $t_{x_i} : T_{x_i} \to \Sigma$

a labeled tree such that T_{x_i} and T_{x_j} are mutually disjoint for $t_{x_i} \neq t_{x_j}$.

Example 9 *Consider the program P 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. The first rule states a sufficient condition on the trust of some C: if different C_1 and C_2 are doing trustworthy business *with C ($t\_bus(C, C_1), t\_bus(C, C_2)$) and C has the ISO 9000 quality certificate ($qual(C, iso9000)$), then C can be trusted as well ($trust(C)$). Moreover, using the minimality of open answer sets, this single rule also expresses that in order for C to be trusted it should be doing trustworthy business with different companies and have the ISO 9000 quality label.* [1] *The constraint encodes the inherent property of t_bus (doing trustworthy business) that if C is doing trustworthy business with D, then D must be a trusted company.*

An answer set, e.g.,

$$
\begin{aligned}
M = \{&trust(x_1), t\_bus(x_1, x_2), t\_bus(x_1, x_3), \\
&\qquad qual(x_1, iso9000), trust(x_2), \ldots\}
\end{aligned}
$$

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 Fig. 1: we call it a forest-model. The forest in Fig. 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 $x \in \mathcal{H} \backslash \mathcal{H}_P$, 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 is still finite. In particular, we have that the roots of the trees in a forest-model may be arbitrarily interconnected. To be formally correct, the forest

[1] Note that adding extra rules with *trust* as the head predicate may change the meaning of trust, i.e., the body of the current rule is not necessarily applied (one could apply the body of an added rule). This differs from other Knowledge Representation formalisms such as Description Logics, where one can express modular sufficient and necessary conditions (by equivalence axioms). Such a modular expression does not seem to be possible with (open) answer set programming.

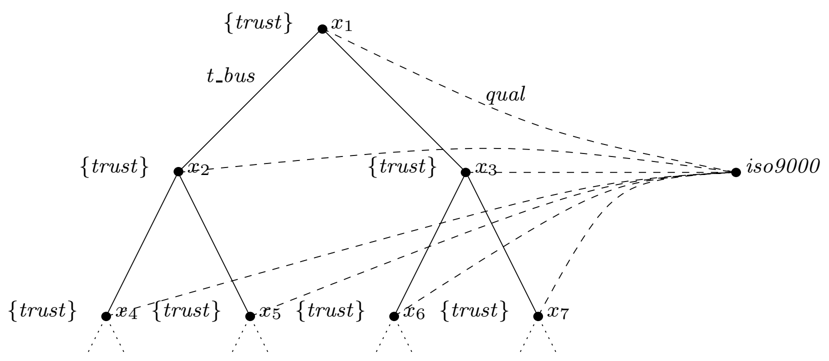

Fig. 1. Forest-Model

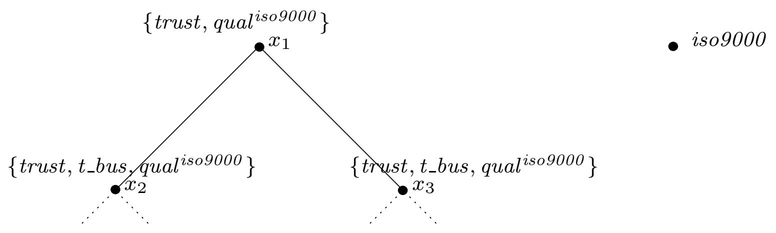
$\{trust, qual^{iso9000}\}$
x_1
iso9000
$\{trust, t\_bus, qual^{iso9000}\}$
x_2
$\{trust, t\_bus, qual^{iso9000}\}$
x_3

Fig. 2. Formal Forest-Model

should not have any labeled edges; we solve this by keeping the label on an edge from x to y in the label of y, and assume that binary predicates in labels refer to edge labels from the predecessor node to the current node, e.g., for $t\_bus(x_1, x_2)$ we keep t_bus in the label of x_2.

Definition 10 *A $p \in upreds(P)$ is **forest-satisfiable** w.r.t. P if there exists an open answer set $(\mathcal{H}, M)$ and a forest $\mathsf{F} = \{t_\varepsilon\} \cup \{t_a \mid a \in \mathcal{H}_P\}$ where the $t_x : \mathcal{H}_x \equiv dom(t_x) \to 2^{preds(P) \cup \{f^a \mid a \in \mathcal{H}_P \wedge f \in bpreds(P)\}}$ are labeled trees with bounded arity such that $\mathcal{H} = \cup_x \mathcal{H}_x$ and $p \in t_\varepsilon(\varepsilon)$. Furthermore, $z \cdot i \in \mathcal{H}_x$, $i > 0$, iff there is some $f(z, z \cdot i) \in M$. For $y \in \mathcal{H}_x$, $q \in upreds(P)$, $f \in bpreds(P)$, we have that*

- *$q(y) \in M$ iff $q \in t_x(y)$, and*
- *$f(y, u) \in M$ iff $(u = y \cdot i \wedge f \in t_x(u)) \vee (u \in \mathcal{H}_P \wedge f^u \in t_x(y))$.*

*We call $(\mathcal{H}, M)$ a **forest-model** and a DLP P has the **forest-model property** if the following property holds: if $p \in upreds(P)$ is satisfiable w.r.t. P then p is forest-satisfiable w.r.t. P. The **label** of a node $z \in \mathcal{H}_x$ is $\mathcal{L}(z) = \{q \mid q \in t_x(z), q \in upreds(P)\}$; for nodes z and u we have that $z < u$ if z is some prefix of u, $\leq$ is defined as usual.*

Example 11 *The forest-model of Example 9, drawn according to Definition 10, is then as in Fig. 2.*

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 class of programs with this forest-model property are *forest logic programs* (FoLPs).

Definition 12 *A FoLP 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*** $a(s) \leftarrow \beta(s), \cup_m \gamma_m(s, t_m), \cup_m \delta_m(t_m), \cup_{i \neq j} t_i \neq t_j$*, 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*.*

We 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.

The program in Example 9 is a FoLP, while the seq-rules from Subsection 2.2 are not FoLP rules, which is consistent with the undecidability of the domino simulation and the decidability of (local) FoLPs, cf. infra. Intuitively, the syntactical restrictions on the rules in a FoLP are designed to ensure the forest-model property, and, to a lesser extent, the bounded finite model property (cf. infra), while ensuring a high degree of expressiveness, e.g., to simulate expressive DLs, see Section 4. E.g., $q(s) \leftarrow not\ f(s, t), \neg q(t)$ is not allowed, since one cannot transform an answer set to a forest-model: assuming $\neg q$ is free, we have that $(\{x, y\}, \{q(x), \neg q(y)\})$ is an answer set, however, it is impossible to make a tree out of this, since we need at least two domain elements, but we do not have a binary predicate to connect them. A similar reason makes $q(s) \leftarrow \neg q(t)$ impossible if t is variable. However, when t is a constant, one does not need an explicit connection between the s-node and t-node since t is the root of its own tree, and thus not part of the tree for s. The latter implies that $q(X) \leftarrow f(X, Y), p(Y), e(a)$ for a constant a is allowed.

Moreover, $f(X, Y) \leftarrow v(X)$ is not allowed, since this may impose connections between x and y without y being a successor of x, $f(X, a) \leftarrow v(X)$ for a constant a on the other hand is allowed. 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].

We can ease the syntactical restrictions on FoLPs by allowing for more general bodies, e.g., by unfolding them, resulting in bodies with a tree-like structure. Complicated constraints $\leftarrow \beta$ can be simulated by a unary rule $a(s) \leftarrow \beta$ and a constraint $\leftarrow a(s)$.

A unary rule $r : a(s) \leftarrow \beta(s), \gamma_m(s, t_m), \delta_m(t_m), t_i \neq t_j$ is a *live* rule if there is a

$\gamma_m \neq \emptyset$ with t_m a variable. A unary predicate a is live if there is a live rule r with a in head(r) and a is not free. The intuition behind a live predicate a is that a new individual y might need to be introduced in order to make $a(x)$ true for an existing x. We denote the set of live predicates for a program P with $live(P)$. A ***degree*** for the liveliness of a rule r, i.e., how many new individuals might need to be introduced to make the head true, is $degree(r) = |\{m \mid \gamma_m \neq \emptyset \wedge t_m \text{ a variable}\}|$. The degree of a live predicate a in P is $degree(a) = \max\{degree(r) \mid a \in \text{head}(r)\}$. E.g., if we only have a rule $r : a(X) \leftarrow f(X, Y_1), g(X, c)$ then a is live and $degree(r) = degree(a) = 1$.

FoLPs indeed have the forest-model property.

Theorem 13 *Forest logic programs have the forest-model property.*

PROOF. Take a FoLP P and $p \in upreds(P)$ s.t. p is satisfiable, i.e., there exists an open answer set $(\mathcal{H}, M)$ with $p(u) \in M$. Let $n = \sum_{a \in live(P)} degree(a)$, i.e., the sum of the degrees of the live predicates. We will define $\theta_x : \{x\} \cdot \{1, \ldots, n\}^* \to \mathcal{H}$ as functions from the full tree with branching n and root $x \in \{\varepsilon\} \cup \mathcal{H}_P$ if $u \notin \mathcal{H}_P$ and $x \in \mathcal{H}_P$ else. The labeled trees $t_x : \text{dom}(\theta_x) \to 2^{preds(P) \cup \{f^a \mid a \in \mathcal{H}_P \wedge f \in bpreds(P)\}}$ are then defined by $t_x(z \cdot i) = \{q \mid q(\theta_x(z \cdot i)) \in M\} \cup \{f \mid f(\theta_x(z), \theta_x(z \cdot i)) \in M\} \cup \{f^a \mid f(\theta_x(z \cdot i), a) \in M\}$.

Initially, we assume $\text{dom}(\theta_x) = \emptyset$, i.e., θ_x is not defined anywhere. The function θ_x is constructed as follows: take $\theta_x(x) = x$ if $x \neq \varepsilon$ and else $\theta_x(x) = u \in \mathcal{H} \backslash \mathcal{H}_P$, and assume we have already considered, as in [54], every member of $\{x\} \cdot \{1, \ldots, n\}^k$, as well as $z \cdot 1, \ldots, z \cdot (m-1)$ for $z \in \{x\} \cup \{1, \ldots, n\}^k$ and $z \in \text{dom}(\theta_x)$. For every live $q \in t_x(z)$, we have that $q(\theta_x(z)) \in M$ and $q(\theta_x(z)) \in T^n$, and since M is an answer set we have that there is a $q(\theta_x(z)) \leftarrow \beta^+(\theta_x(z)), \gamma_m^+(\theta_x(z), y_m), \delta_m^+(y_m)$, with the body true in M and in T^{n-1}. If for all i either $\gamma_i = \emptyset$ or $y_i \in \mathcal{H}_P$, i.e., we do not have a live rule, then we continue with the next $q \in t_x(z)$, otherwise, for i, $\gamma_i \neq \emptyset$ and $y_i \notin \mathcal{H}_P$, if there is a $z_j \in \{z \cdot 1, \ldots, z \cdot (m-1)\}$ with $\theta(z_j) = y_i$ then θ remains undefined on $z \cdot (m + i)$, otherwise $\theta(z \cdot (m + i)) = y_i$. Note that $t_x(z) \neq \emptyset$, since θ_x is defined on z.

One can show that $(\cup_x \text{dom}(t_x), \{q(z) \mid q \in t_x(z)\} \cup \{f(z, z \cdot i) \mid f \in t_x(z \cdot i)\} \cup \{f(z, a) \mid f^a \in t_x(z)\})$ is an open answer set of P such that $\mathsf{F} = \cup_x \{t_x\}$ is a forest satisfying the conditions from Definition 10. $\square$

3.2 Bounded Finite Model Property

Satisfiability checking w.r.t. the FoLPs in [31] was shown to be decidable by a reduction to two-way alternating tree automata [54]. However, the current definition of FoLPs includes constants, which were not allowed in [31], such that the automata reduction cannot be readily applied. Moreover,while automata provide an elegant characterization, there are few implementations available, e.g., [32] implements a specific type, looping alternating automata, using a translation to description logics.

An alternative approach is to identify a particular class of FoLPs, satisfying the *local model property*, that allow for satisfiability checking with existing answer set solvers such as DLV [41] or SMODELS [50], since they have the *bounded finite model property*. 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.

FoLPs with the local model property are such that they are satisfiable by forest-models where the presence of each literal in such a model is locally motivated by the involved node, a successor of the node, and/or a constant.

Definition 14 *Let P be a FoLP and for a literal l, $\mathcal{H}_{S(l)}$ the domain elements in $S(l)$, the support of l. A forest-model $(\mathcal{H}, M)$ of P is* ***locally supported*** *if*
$\forall l = q(x) \in M \vee l = f(x, y) \in M \cdot$
$(\mathcal{H}_{S(l)} \subseteq \{x, xi\} \cup \mathcal{H}_P) \wedge (\forall f(z, a) \in S(l), a \in \mathcal{H}_P \cdot z \neq xi)$*, i.e., the support for a literal involves only the domain element x under consideration, successors $x{\cdot}i$, or constants. $p \in upreds(P)$ 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 $\mathcal{H}$. A FoLP 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.*

In the above definition, the extra condition, $\forall f(z, a) \in S(l), a \in \mathcal{H}_P {\cdot} z \neq xi$, makes sure that constants do not sneak around the locality by providing support for a literal at x via xi. As we will indicate below, cutting a tree at an xi may remove $f(xi, a)$. If $f(xi, a)$ were then in the support of a literal in x, that literal would end up without support in the cut tree.

Example 15 *Take the program from Example 9. The forest-model in Fig. 1 is a locally supported forest-model, e.g., a support*

$$S(trust(x_1)) = \{trust(x_1), t\_bus(x_1, x_2), t\_bus(x_1, x_3), qual(x_1, iso9000)\}$$

such that no other domain elements than the domain element under consideration, its immediate successors or constants motivate the presence of a literal.

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 the blocking technique for DL tableaux [5], but the minimality of answer sets makes it non-trivial and only valid for FoLPs with the local model property, as we indicate below.

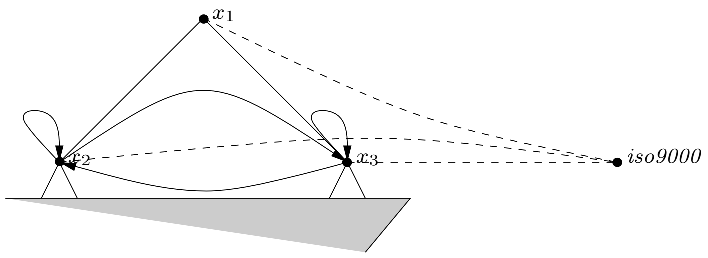

Fig. 3. Bounded Finite Model

Example 16 *Considering the forest-model in Fig. 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 that $t\_bus(x_i, x_2)$, $t\_bus(x_i, x_3)$, and $qual(x_i, iso9000)$ for $i = 2$ and $i = 3$, resulting in the answer set depicted in Fig. 3.*

Formally, a FoLP P has the *bounded finite model property* if the following holds: if $p \in upreds(P)$ is satisfiable w.r.t. P then there is a finite answer set $(\mathcal{H}, M)$ of P and a nonnegative integer k, defined as a function of P, such that $p(x) \in M$ and $|\mathcal{H}| < k$. The bounded finite model property is similar to the *small model property* found in the temporal logic CTL [19] where a CTL formula is satisfiable iff it is satisfiable by a model that has a number of states at most exponential in the length of the formula.

Theorem 17 *Let P be a FoLP with the local model property. Then, P has the bounded finite model property.*

PROOF. Assume p is satisfiable w.r.t. P. Since P has the local model property, there is a locally supported forest-model $(\mathcal{H}, M)$ with $p(\varepsilon) \in M$. $\mathcal{H}$ is a multi-set of trees $\cup_x \mathcal{H}_x$ with roots x, for $x \in \{\varepsilon\} \cup \mathcal{H}_P$, where possibly ε is some $a \in \mathcal{H}_P$. Let m be the number of different labels in the forest-model. For a path $\mathcal{P}$ of length at least $m + 1$ in a $\mathcal{H}_x$, define $z_\mathcal{P} \in \mathcal{H}_x$ as the minimal node (w.r.t. the prefix relation $<$) s.t. $\exists y < z_\mathcal{P} \cdot y \notin \mathcal{H}_P \wedge \mathcal{L}(y) = \mathcal{L}(z_\mathcal{P})$. Denote this unique y with $\overline{z}_\mathcal{P}$. Since we have a finite number m of different labels, we must have that for every path $\mathcal{P}$ of length m there are two nodes with the same label, moreover, in the worst case we only need a path of length $m + 1$ to make sure that $\overline{z}_\mathcal{P}$ is not a constant. Note that $z_\mathcal{P}$ nor $\overline{z}_\mathcal{P}$ can be a constant, since constants may be introduced by rules containing no variables in the head, which, consequently, cannot be used to motivate the presence of literals at anonymous nodes: it might be that a rule $t(a) \leftarrow$ introduces t in the label of some constant 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.

Define $\mathcal{H}'_x = \{z \in \mathcal{H}_x \mid (z \in \mathcal{P} \wedge |\mathcal{P}| > m \Rightarrow z \leq z_\mathcal{P}\}$, i.e., cut the tree $\mathcal{H}_x$

at $z_{\mathcal{P}}$ for every path $\mathcal{P}$ that has length at least $m+1$, and let $\mathcal{H}' = \cup_x \mathcal{H}'_x$. Define $M' = \{q(z) \mid z \in \mathcal{H}', q(z) \in M\} \cup \{f(z,y) \mid z \in \mathcal{P} \Rightarrow z < z_{\mathcal{P}}, f(z,y) \in M\} \cup \{f(z_{\mathcal{P}}, y) \mid f(\overline{z}_{\mathcal{P}}, y) \in M\}$.

From Theorem 13, we have that the branching of a $\mathcal{H}_x$ is at most

$$n \equiv \sum_{a \in live(P)} degree(a) \ ,$$

such that the number of nodes in $\mathcal{H}'_x$ is at most $\sum_{i=0}^{m+1} n^i$. We have that $\mathcal{H}'$ contains at most $c+1$ trees $\mathcal{H}'_x$, where $c \equiv |\mathcal{H}_P|$, such that the cardinality of $\mathcal{H}'$ is at most $(c+1)\sum_{i=0}^{m+1} n^i$. Note that $m \leq 2^u$ with $u = |upreds(P)|$ such that the cardinality of $\mathcal{H}'$ is at most

$$k \equiv (c+1) \sum_{i=0}^{2^u+1} n^i \ , \tag{1}$$

where k is calculated as a function of P only.

Further note that $p(\varepsilon) \in M'$, such that it only remains to show that $(\mathcal{H}', M')$ is an answer set. $\square$

The local model property is a necessary property, i.e., the described cutting technique does not work for arbitrary FoLPs.

Example 18 *Consider rules $a(X) \leftarrow f(X,Y), a(Y)$ and $a(X) \leftarrow b(X)$ with b and f free predicates. A forest-model of this program is*

$$\{a(\varepsilon), f(\varepsilon,1), a(1), f(1,11), a(11), b(11)\} \ .$$

Since ε and 1 have the same label we cut the tree at 1. In the resulting structure $\{a(\varepsilon), f(\varepsilon,1), a(1), f(1,1)\}$, $a(\varepsilon)$ nor $a(1)$ are motivated, as $b(11)$ is no longer present. The resulting structure is thus not minimal.

FoLPs with the local model property 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.

Satisfiability checking w.r.t. FoLPs with the local model property can then be done by standard answer set solvers. Intuitively, we introduce a large enough number of constants, such that every bounded finite model, that is guaranteed to exist by the local model property, can be mapped to these constants.

Theorem 19 *Let P be a FoLP with the local model property. $p \in upreds(P)$ is satisfiable w.r.t. P iff there is a $0 \leq h \leq k$ and an answer set M of $\psi_h(P)$ containing a p-atom, where k is as in (1) and $\psi_h(P) \equiv P \cup \{cte(x_i) \leftarrow \ \mid 1 \leq i \leq h\}$.*

PROOF. For the "only if" direction, assume p is satisfiable w.r.t. P, such that, by Theorem 17, there is an open answer set $(\mathcal{H}', M')$ of P, with $|\mathcal{H}'| \leq k$. Define $h \equiv |\mathcal{H}'| - |cts(P)|$, i.e., the number of anonymous elements in $\mathcal{H}'$. Define a bijection $F : \mathcal{H}' \to \mathcal{H}_{\psi_h(P)}$ such that $F(a) = a$ for $a \in \mathcal{H}_P$. Define $M \equiv \{a(F(x)) \mid a(x) \in M'\} \cup \{f(F(x), F(y)) \mid f(x, y) \in M'\} \cup \{cte(x_i) \mid 1 \leq i \leq h\}$. Intuitively, we identify the forest $\mathcal{H}'$ with the constants in $\psi_h(P)$. One can show that M is an answer set of $\psi_h(P)$.

For the "if" direction, assume there exists an answer set M of $\psi_h(P)$ containing a p-atom. Define $\mathcal{H}' \equiv \mathcal{H}_{\psi_h(P)}$, one can show that $(\mathcal{H}', M' \equiv M \backslash \{cte(x_i) \mid 1 \leq i \leq h\})$ is an open answer set of P. $\square$

Note that standard answer set solvers such as DLV or SMODELS do not allow negation as failure in the head, but this can be solved with the transformation of such programs to programs without *not* in the head [39] .

The local model property is a semantic property which makes Theorem 19 non-trivial to use. However, a particular syntactic class of FoLPs that have the local model property are *local FoLPs*.

Definition 20 *A **local FoLP** is a FoLP 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 $r : b(s) \leftarrow$ body(r), body$(r)^+ = \emptyset$.

Example 21 *The program from Example 9 is a local FoLP while the program from Example 18 is not. Note that the latter example does not have the local model property either; in Example 23, we give a non-local program that does have the local model property.*

Intuitively, local FoLPs can motivate an $a(s)$ ($f(s, t)$) in an answer set, by descending 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 to going further down the tree, but whilst doing this one does not need to remember which literals need to be proved above in the tree. In a way a local FoLP has limited memory: it only remembers the previous (predecessor) state. A similar intuition applies to algorithms that check satisfiability of certain modal logics. E.g., [27] (Theorem 6.11) defines a PSPACE algorithm for checking satisfiability of the modal logic K_n, based on a marking that assigns *satisfiable* to a state depending solely on the label of that state and the marking of the successors. Such an algorithm makes the decision to mark a state *satisfiable* in a local way. Analogously, predicates in the label of a node in a forest-model are motivated by looking at the label of the node and labels of the successor nodes. Note that the algorithm in [27] is an extension for K_n (a modal logic with n *agents*) of the modal logic K (for one agent) in [40].

Theorem 22 *Every forest-model of a local FoLP is locally supported, and, as a consequence, local FoLPs have the local model property.*

There are FoLPs with the local model property that are not local FoLPs, making the syntactical restriction less expressive than the semantical characterization.

Example 23 *Take the FoLP*

$$\begin{aligned} a(X) &\leftarrow f(X, Y), b(Y) \\ a(X) &\leftarrow c(X) \\ b(X) &\leftarrow c(X) \\ &\leftarrow b(X) \end{aligned}$$

with f and c free. This program is not local as b in the first rule does not satisfy the necessary conditions. However, every predicate is satisfiable by a locally supported forest-model such that the program has the local model property. Intuitively, the first rule, which is problematic for syntactical locality, will never be applicable in an open answer set since the constraint $\leftarrow b(X)$ prohibits this. The example suggests that finding a syntactical characterization that corresponds to the semantical characterization (local iff local model property) is not trivial: the local supportedness of the forest-model is guaranteed by non-applicability of certain rules, which seems hard to enforce syntactically in general.

3.3 Complexity

Let P be a FoLP. We verify the complexity of checking whether there exists an answer set M of $\psi_h(P)$ for some $0 \leq h \leq k$ where k and $\psi_h(P)$ are as in Theorem 19. We distinguish between two cases:

- If FoLP rules have a degree bounded by m, independent of a particular FoLP, then the size of $ground(\psi_h(P))$ is polynomial in the size of $\psi_h(P)$, since every rule in $\psi_h(P)$ introduces at most $\mathcal{O}(|\mathcal{H}_{\psi_h(P)}|^{m+1})$ rules in $ground(\psi_h(P))$. Indeed, each FoLP rule then contains at most $m+1$ variables, each of which can be instantiated with a constant from $\psi_h(P)$. Since checking whether there exists an answer set M of $\psi_h(P)$ is in NP in the size of $ground(\psi_h(P))$ [14,7], we have that checking whether there exists an answer set M of $\psi_h(P)$ is in NP in the size of $\psi_h(P)$ as well.

- If the degree is not bounded, we use a result from [16] to state that checking whether M is an answer of $\psi_h(P)$ is in Σ_2^{P} w.r.t. the size of $\psi_h(P)$.[2] Indeed, the arities of predicates in $\psi_h(P)$ are bounded by 2 since FoLPs allow only for unary and binary predicates.

Thus, for a fixed h, checking whether $\psi_h(P)$ has an answer set is in NP for a FoLP with bounded degree and in Σ_2^{P} in general.

Satisfiability checking of a predicate w.r.t. P can then be done by starting with $h = 0$ and checking whether $\psi_h(P)$ has an answer set containing a p-atom. If this is the case, we are done (by Theorem 19), otherwise, we repeat the check for $h = 1$, and so on. If finally $h = k$ has been checked, i.e., $\psi_h(P)$ had no answer sets containing a p-atom, one can conclude, by Theorem 19, that the predicate is not satisfiable. This procedure thus involves at most $k + 1$ calls to an NP oracle for FoLPs with bounded degree or to an Σ_2^{P} oracle in general.

We have that

$$k = (c+1)\sum_{i=0}^{2^u+1} n^i = (c+1)\frac{(1-n^{2^u+2})}{(1-n)} \ ,$$

with $u = |upreds(P)|$, $c = |cts(P)|$, and n the rank of P such that k is double exponential in the size of P and the above procedure to check satisfiability runs in $2\text{-}\textsc{exptime}^{\mathrm{NP}}$ for FoLPs with bounded degree and the local model property or in $2\text{-}\textsc{exptime}^{\Sigma_2^{\mathrm{P}}}$ for arbitrary FoLPs with the local model property.

Theorem 24 *Let P be a FoLP with the local model property. Satisfiability checking w.r.t. P is in $2\text{-}\textsc{exptime}^{\Sigma_2^{\mathrm{P}}}$ for a non-bounded degree of FoLP rules or in $2\text{-}\textsc{exptime}^{\mathrm{NP}}$ otherwise.*

3.4 Extended Forest Logic Programs

Consider a FoLP 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 date cheaters.

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

[2] Recall that $\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$.

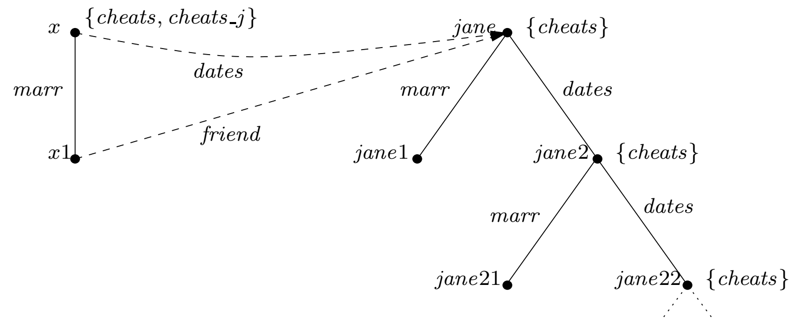

Fig. 4. Forest-Model

with $marr$, $friend$ and $dates$ free predicates. An (infinite) answer set of this program that satisfies $cheats\_j$ is depicted in Fig. 4. 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. In many cases, there is interesting knowledge that cannot be captured within the rather strict tree format of FoLP rules. For example, in addition, 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, but relates the three constants in an arbitrary graph-like manner. We extend FoLPs by allowing for a component with arbitrary DLP rules that may only be grounded with the combined program's constants.

Definition 25 *An* ***extended forest logic program (EFoLP)*** *P is a pair (Q, R) where Q is a FoLP and R is a finite DLP. We denote Q with $clp(P)$ and R with $e(P)$. An* ***EFoLP answer set*** *of (Q, R) is an open answer set of $Q \cup R_{\mathcal{H}(Q \cup R)}$. Satisfiability checking and query answering w.r.t. (Q, R) are modified accordingly.*

To avoid confusion with EFoLP answer sets and open answer sets, we assume an EFoLP P is a FoLP Q extended with a ground DLP R, i.e., $P = Q \cup R$, under an open answer set semantics. It is easy to see that the EFoLP answer set semantics of an EFoLP can be reduced to the open answer set semantics of a FoLP with an arbitrary ground part.

Note that $e(P)$ can be full-fledged DLP, i.e., with negation as failure. Moreover, predicates in $e(P)$ may be defined in the FoLP $clp(P)$, as is the case for $marr$, $dates$ and $cheats$. Vice versa, we may have predicates appearing in the FoLP part that are defined in the ground rule part, e.g., $dislikes$ could appear in the FoLP part as a $dislikes(X, Y)$ literal.

EFoLPs 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 EFoLP P can ever introduce. Proofs in this subsection are adaptations from

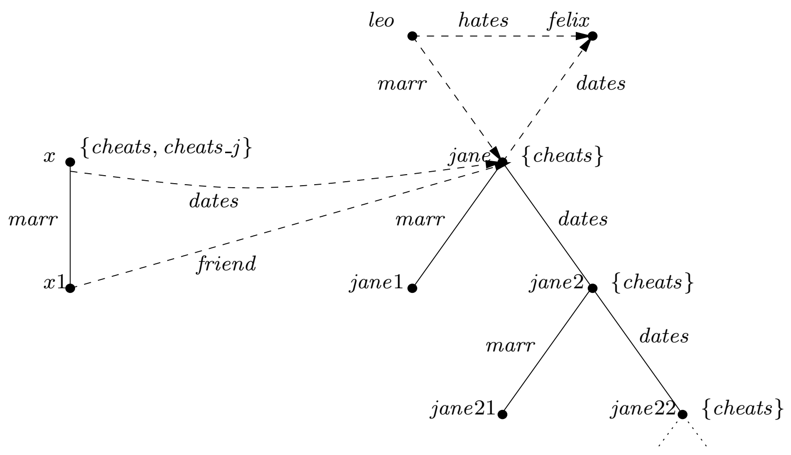

Fig. 5. Forest-Model of the EFoLP

their FoLP counterparts and have been omitted.

Theorem 26 *Extended forest logic programs have the forest-model property.*

The forest-model of the cheats example is depicted in Fig. 5. The cutting of infinite answer sets to finite ones, as defined in Subsection 3.2, cannot be applied to arbitrary EFoLPs. As in the FoLP case, we need a local model property. Unfortunately, the local model property as defined for FoLPs will not do. 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 also involve successors of constants whereas the local model property definition for FoLPs allows only the constants themselves in the support.

Although the local model property for FoLPs is not suitable, it can be safely relaxed by allowing also successors of constants in the support. Indeed, 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 the successors of constants remain unmodified in the cut forest.

Definition 27 *A forest-model* $(\mathcal{H}, M)$ *of an EFoLP* P *is* ***locally supported*** *if*
$\forall l = q(x) \in M \vee l = f(x, y) \in M \cdot$
$(\mathcal{H}_{S(l)} \subseteq \{x, xi\} \cup \{a, ai \mid a \in \mathcal{H}_P\})\wedge$
$(\forall f(z, a) \in S(l), a \in \mathcal{H}_P \cdot z \neq xi)$, $p \in upreds(P)$ *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 EFoLP* 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.*

EFoLPs with the local model property then have the desired bounded finite model property.

Theorem 28 *Let* P *be an EFoLP with the local model property. Then,* P *has the bounded finite model property.*

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

Theorem 29 *Let* P *be an EFoLP with the local model property.* $p \in upreds(P)$ *is satisfiable w.r.t.* P *iff there is a* $0 \leq h \leq k$ *and an answer set* M *of* $\psi_h(P)$ *containing a* p*-atom, where* k *and* $\psi_h(P)$ *are as in Theorem 19.*

The other direction is trivial: there is a normal answer set M of a program P containing a $p(a) \in \mathcal{H}_P$ iff p is satisfiable w.r.t. to the EFoLP $(\emptyset, P)$. Indeed, by definition of EFoLPs, the second component in the pair has a normal answer set semantics. By [14,7], the normal answer set semantics for DLPs is NEXPTIME-complete. Furthermore, $(\emptyset, P)$ has the local model property such that we have the following lower complexity bound.

Theorem 30 *Let* P *be an EFoLP with the local model property. Satisfiability checking w.r.t.* P *is* NEXPTIME*-hard.*

A lower EXPTIME bound for reasoning with FoLPs will be established in Section 4. Similar to the complexity upper bound for FoLPs with the local model property, one can deduce the following upper bounds for EFoLPs with the local model property (where extra complexity is due to the unbounded grounding of the arbitrary rule part).

Theorem 31 *Let* P *be an EFoLP with the local model property. Satisfiability checking w.r.t.* P *is in* 2-EXPTIME$^{\text{NEXPTIME}}$.

As was the case for FoLPs, the local model property for EFoLPs is a semantical characterization, which makes it non-trivial to recognize EFoLPs satisfying this property. We identify a class of EFoLPs, based on their syntactic structure, that have the local model property.

Definition 32 *A* ***local EFoLP*** P *is an EFoLP where* $clp(P)$ *is a local FoLP.*

Local EFoLPs have the local model property, i.e., the arbitrary rules have no influence on the locality.

Theorem 33 *Local EFoLPs have the local model property.*

4 Nonmonotonic Ontological and Rule-based Reasoning with Extended Forest Logic Programs

In Subsection 4.1, we simulate reasoning in an expressive DL with FoLP; Subsection 4.2 shows that the extension of this DL with DL-safe rules can be simulated by EFoLP, and discusses some of the advantages of EFoLPs for representing and reasoning with conceptual and rule-based knowledge.

4.1 Ontological Reasoning with FoLPs

Description logics (DLs) [5] 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 [9]. 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.

$\mathcal{ALCHOQ}(\sqcup, \sqcap)$ is a particular DL with syntax and semantics as in Table 1; concept names A are the base concept expressions, P is a role name, establishing the base role expression, and o is an individual. D and E are arbitrary concept expressions, and R and S are arbitrary role expressions. Individuals are interpreted as elements in $\Delta^{\mathcal{I}}$, concept expressions as subsets of $\Delta^{\mathcal{I}}$ and role expressions as binary relations on $\Delta^{\mathcal{I}}$. DLs are named according to their constructs: $\mathcal{AL}$ is the basic DL [49], 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.

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. Note that OWL does not have the unique name assumption [51], 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.

For concept expressions D and E, *terminological axioms* $D \sqsubseteq E$ are satisfied by an interpretation $\mathcal{I}$ 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 $C^{\mathcal{I}} \neq \emptyset$.

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	$\{o\}^{\mathcal{I}} =$	$\{o^{\mathcal{I}}\} \subseteq \Delta^{\mathcal{I}}$
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	$(\leq n\ R.D)^{\mathcal{I}} =$	$\{x \mid \#\{y \mid (x, y) \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\} \leq n\}$
	$(\geq n\ R.D)^{\mathcal{I}} =$	$\{x \mid \#\{y \mid (x, y) \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\} \geq n\}$

As an example, the human resources department has 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 they are the boss of at least three $Workers$. This corresponds to the following $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base Σ:

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

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

We can rewrite Σ as an equivalent FoLP P. The axioms in Σ correspond to the constraints

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

in P, 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 $(\mathit{Manag} \sqcup \mathit{Workers} \sqcup \{\mathit{john}\})(x)$. Those constraints are the kernel of the translation; we still need, however, to simulate the DL semantics by rules that define the different DL constructs.

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

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

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

The predicate $(\exists \mathit{boss}.\mathit{Manag})$ is defined by

$$(\exists \mathit{boss}.\mathit{Manag})(X) \leftarrow \mathit{boss}(X, Y), \mathit{Manag}(Y)$$

such that, if the literal $(\exists \mathit{boss}.\mathit{Manag})(x)$ is in the answer set, there is a y such that $\mathit{boss}(x, y)$ and $\mathit{Manag}(y)$ are in the answer set and vice versa.

The predicate $((\forall \mathit{t\_orders}.\mathit{Manag}) \sqcap (\geq 3\ \mathit{boss}.\mathit{Workers}))$ is defined by

$$\begin{aligned}
&((\forall \mathit{t\_orders}.\mathit{Manag}) \sqcap (\geq 3\ \mathit{boss}.\mathit{Workers}))(X) \leftarrow\\
&\qquad\qquad (\forall \mathit{t\_orders}.\mathit{Manag})(X), (\geq 3\ \mathit{boss}.\mathit{Workers})(X)
\end{aligned}$$

and the body predicates by the rules

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

and

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

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.

$$Workers(X) \vee not\ Workers(X) \leftarrow$$
$$Personnel(X) \vee not\ Personnel(X) \leftarrow$$
$$Manag(X) \vee not\ Manag(X) \leftarrow$$
$$boss(X, Y) \vee not\ boss(X, Y) \leftarrow$$
$$t\_orders(X, Y) \vee not\ t\_orders(X, Y) \leftarrow$$

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 $M = \{C(x) \mid C \in upreds(P), x \in C^{\mathcal{I}}\} \cup \{R(x,y) \mid R \in bpreds(P), (x,y) \in R^{\mathcal{I}}\}$, 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 that $D(R) \in clos(C, \Sigma)$,
- for every D in $clos(C, \Sigma)$, we distinguish the following cases:

$$D = \neg D_1 \quad \Rightarrow \quad D_1 \in clos(C, \Sigma)$$
$$D = D_1 \sqcup D_2 \quad \Rightarrow \quad \{D_1, D_2\} \subseteq clos(C, \Sigma)$$
$$D = D_1 \sqcap D_2 \quad \Rightarrow \quad \{D_1, D_2\} \subseteq clos(C, \Sigma)$$
$$D = \exists R.D_1 \quad \Rightarrow \quad \{R, D_1\} \subseteq clos(C, \Sigma)$$
$$D = \forall R.D_1 \quad \Rightarrow \quad \{D_1, \exists R.\neg D_1\} \subseteq clos(C, \Sigma)$$
$$D = (\leq n\ Q.D_1) \quad \Rightarrow \quad \{(\geq n+1\ Q.D_1)\} \subseteq clos(C, \Sigma)$$
$$D = (\geq n\ Q.D_1) \quad \Rightarrow \quad \{Q, D_1\} \subseteq clos(C, \Sigma)$$

- 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 FoLP $\Phi(C, \Sigma)$ 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.

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

Table 2
FoLP Translation $\Phi(C, \Sigma)$

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

Theorem 34 *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 $o^{\mathcal{I}}$ 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 $M = \{C(x) \mid x \in C^{\mathcal{I}}, C \in \mathit{clos}(C, \Sigma)\} \cup \{R(x, y) \mid (x, y) \in R^{\mathcal{I}}, R \in \mathit{clos}(C, \Sigma)\}$. 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 $C^{\mathcal{I}} \neq \emptyset$. $\square$

Note that, in general, the resulting FoLP $\Phi(C, \Sigma)$ is not local: $(\exists R.(A \sqcap B))$ is translated as 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 acyclic. 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 $(\exists R.(A \sqcap B))(X) \leftarrow R(X, Y), \mathit{not}\ (A \sqcap B)'(Y)$; $(A \sqcap B)'(X) \leftarrow \mathit{not}\ (A \sqcap B)(X)$: we negate $(A \sqcap B)(Y)$ twice. The resulting FoLP is now local and satisfiability checking w.r.t. $\Phi(C, \Sigma)$ can be reduced to this replacement, as a consequence of Theorem 6.

From the reduction of reasoning with $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ to reasoning with local FoLPs, we can deduce a lower complexity bound for reasoning with the latter. Indeed, since satisfiability checking of the sublanguage $\mathcal{AL}$ w.r.t. a set of axioms is EXPTIME-complete [5], we have the following theorem.

Theorem 35 *Let P be a FoLP with the local model property. Satisfiability checking w.r.t. P is* EXPTIME*-hard.*

The $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ simulation shows the feasibility of Semantic Web reasoning with FoLPs, as $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ is an expressive DL related to the OWL DL ontology language. Formally, OWL DL corresponds to the DL $\mathcal{SHOIN}(\mathbf{D})$ [37], which differs from $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ in that $\mathcal{SHOIN}(\mathbf{D})$ additionally allows for inverted roles ($\mathcal{I}$), data types ($\mathbf{D}$) and transitivity of roles (which distinguishes $\mathcal{S}$ from $\mathcal{ALC}$). However, $\mathcal{SHOIN}(\mathbf{D})$, and thus OWL DL, does not support qualified number restrictions, i.e., it only allows for unqualified number restrictions such as $(\geq n\ R)$ instead of qualified ones $(\geq n\ R.D)(X)$. Furthermore, $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ adds the role constructs $\sqcup$ and $\sqcap$.

Putting this in perspective, the loss of transitivity in $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ weighs heavier than having qualified number restrictions and role constructors. Indeed, there is actually no reason why OWL DL should not include qualified number restrictions (corresponding to the DL $\mathcal{SHOIQ}(\mathbf{D})$). We needed to omit transitivity in order to be able to translate to EFoLPs with the bounded finite model property. OWL DL does not have this limitation, i.e., there are concept expressions that have only infinite models. Note that adding transitivity to $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ without restricting the allowed roles in qualified number restrictions (they cannot be transitive nor can they have transitive subroles), one immediately has undecidability of reasoning [35]. Further note that OWL DL does not make the unique name assumption, while EFoLPs do. Since the unique name assumption can be asserted in OWL DL, EFoLPs are strictly weaker in this respect.

4.2 Combined Ontological and Rule-based Reasoning with EFoLPs

The ontology layer for the Semantic Web is becoming a reality with languages such as OWL DL, and the rule layer, which provides additional inferencing capabilities on top of DL reasoning, is gaining interest in the Semantic Web community. For example, in [45], 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 [45]

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

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

We introduce DL-safe rules as in [45]. 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., [12], 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 [45] 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 [33] rather than on first-order interpretations. However, both semantics are compatible as indicated in [45]. 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 $\sigma : vars(P) \cup cts(\Sigma, P) \to \Delta^{\mathcal{I}}$ 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 model $\mathcal{I}$ of (Σ, P), the ground atom α is true in $\mathcal{I}$.

In Subsection 4.1, we reduced $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ satisfiability checking to FoLP satisfiability checking. We can reduce query answering w.r.t. $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge bases extended with DL-safe rules to query answering w.r.t. EFoLPs. We first provide some intuition with an example. Take a knowledge base Σ

$$\exists manuf\_in.Co \sqcap \exists has\_price \sqsubseteq Product \ ,$$

expressing that if something is manufactured in some country and it has a price then it is a product ($\exists has\_price$ is shorthand for $\exists has\_price.\top$, where $\top^{\mathcal{I}} \equiv \Delta^{\mathcal{I}}$ for every interpretation $\mathcal{I}$). We have some facts in a DL-safe program P about the world we are considering:

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

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 trivially 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, those companies are competitors [3], $r_1 : competitors(C_1, C_2) \leftarrow Product(P), is\_product\_of(P, C_1),$ $is\_product\_of(P, C_2)$. Note that this is indeed a DL-safe rule since every variable occurs in a $is\_product\_of$ atom, which is a non-DL-atom in the body of the rule. The only DL-atom in the rule is $Product(P)$. A model $\mathcal{I}$ of (Σ, P) is $\mathcal{I} = (\{japan, c_1, c_2, p, x\}, \cdot^{\mathcal{I}})$ [4] with $\cdot^{\mathcal{I}}$: $Co^{\mathcal{I}} = \{japan\}$, $Product^{\mathcal{I}} = \{p\}$, $manuf\_in^{\mathcal{I}} = \{(p, japan)\}$, $has\_price^{\mathcal{I}} = \{(p, x)\}$, $is\_product\_of^{\mathcal{I}} = \{(p, c_1), (p, c_2)\}$, $competitors^{\mathcal{I}} = \{(c_1, c_2)\}$.

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

$$(\exists manuf\_in.Co \sqcap \exists has\_price)(X) \leftarrow (\exists manuf\_in.Co)(X), (\exists has\_price)(X)$$
$$(\exists manuf\_in.Co)(X) \leftarrow manuf\_in(X, Y), Co(Y)$$
$$(\exists has\_price)(X) \leftarrow has\_price(X, Y)$$

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.

$$Product(X) \vee not\ Product(X) \leftarrow$$
$$Co(X) \vee not\ Co(X) \leftarrow$$
$$manuf\_in(X, Y) \vee not\ manuf\_in(X, Y) \leftarrow$$
$$has\_price(X, Y) \vee not\ has\_price(X, Y) \leftarrow$$

This concludes the FoLP part of the translation of (Σ, P). Formally, we define $\Phi(\Sigma)$ as the $\Phi(C, \Sigma)$ from Subsection 4.1 where C is some arbitrary concept from Σ. The arbitrary DLP part of the EFoLP includes the DL-safe rules.

Since DL-safe rules have a first-order interpretation it may be that

$$(c_1, c_2) \in competitors^{\mathcal{I}}$$

for a model $\mathcal{I}$ of (Σ, P) without any justification in $\mathcal{I}$: the body of r_1 in P does not need to be satisfied in order to have $(c_1, c_2) \in competitors^{\mathcal{I}}$. The answer set semantics, however, only deduces $competitors(c_1, c_2)$ in an answer set if the body

[3] 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 [45], however, it is with EFoLPs using $\neq$.

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

of r_1 is satisfied in that answer set, since otherwise the answer set would not be minimal (one could omit $competitors(c_1, c_2)$ and still have an answer set).

To solve this, we introduce for each head a of a DL-safe rule, a rule $a \vee not\ a \leftarrow$, $competitor(C_1, C_2) \vee not\ competitor(C_1, C_2) \leftarrow$, such that one has always a motivation for $competitor(C_1, C_2)$, mimicking the first-order semantics.

Formally, we define $\chi(P)$ for a DL-safe program P as the DLP P with free rules

$$\mathrm{head}(r) \vee not\ \mathrm{head}(r) \leftarrow \ ,$$

for each $r \in P$.

Theorem 36 *For an $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base Σ and a DL-safe program P, we have $(\Sigma, P) \models \alpha$ iff $(\Phi(\Sigma), \chi(P)) \models \alpha$.*

In [45] the DL $\mathcal{SHOIN}(\mathbf{D})$ is considered in the definition of DL-safe rules instead of $\mathcal{ALCHOQ}(\sqcup, \sqcap)$. Decidability of query answering is shown for the DL $\mathcal{SHOIN}$ (i.e., without data types)[5]. Using EFoLPs 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 FoLP part and the 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).

Example 37 *Add a rule to the company example ontology, expressing that if John is not married, he works late at the office:*

$$works\_late(john) \leftarrow not\ married(john)$$

Adding such a rule to our knowledge will have the effect that every open answer set includes the literal $works\_late(john)$, i.e., John always works late. However, consecutively adding the newly acquired knowledge that John is actually married with a rule

$$married(john) \leftarrow$$

will make sure that John never works late in answers to our current knowledge. This type of nonmonotonicity *is one of the main strengths of logic programming paradigms for knowledge representation and is thus useful in Semantic Web reasoning as well; it was, e.g., identified in [13] as one of the requirements on a logic for reasoning on the Web. DLs lack this feature and are* monotonic, *e.g., one could try to translate the above rule as the following DL axiom.*

$$\neg Married \sqcap \{john\} \sqsubseteq Works\_late \sqcap \{john\}$$

[5] Note that the proof of this decidability does not use a reduction to disjunctive Datalog; in order to use such a reduction [45] restricts itself to $\mathcal{SHIQ}(\mathbf{D})$.

However, it is clear that interpretations satisfying this axiom have a choice in making John work later or not, such that adding that John is married would not invalidate any previously concluded facts.

Besides the previously illustrated nonmonotonicity, FoLPs are more articulate than DLs in other aspects.

Example 38 *E.g., representing the knowledge that a team must at least*[6] *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 in order for the rule to correspond to our informal definition of a team we assume no other rules with a head predicate team *exist, i.e., we implicitly use the minimality of open answer sets. This is clearly not ideal. However, using only satisfaction of rules to conclude that, if x is team, then it should satisfy the listed properties, seems impossible to express with (open) answer set programming. Compare the rule with, e.g., the rule for number restrictions in Table 2. In number restrictions $(\geq n\ R.C)$ one indicates that there are more than n R-successors that are of type C, while FoLPs can constrain different successor relationships ($member$ and $leader$) instead of just one (R). Moreover, FoLPs can be very specific about which successors should be different and which ones may be equal (Y_1 may be equal to Y_2, but should be different from Y_3), or to which different types the successors belong ($tech$ and $secret$) instead of one type (C).*

Representing such generalized number restrictions using DLs would be significantly harder while arguably less succinct.

Finally, consider some EFoLP (Q, R) where R is the ground rule

$$f(a, c) \leftarrow f(a, b), f(b, c)$$

Although this rule does not have a tree structure, its groundness suggests that one can replace it by a DL axiom using nominals:

$$\{a\} \sqcap \exists f.(\{b\} \sqcap \exists h.\{c\}) \sqsubseteq \{a\} \sqcap \exists f.\{c\}$$

If $(a, b) \in f^{\mathcal{I}}$ and $(b, c) \in f^{\mathcal{I}}$ for a model $\mathcal{I}$ (and assuming a, b, and c are the elements of $\{a\}^{\mathcal{I}}$, $\{b\}^{\mathcal{I}}$, and $\{c\}^{\mathcal{I}}$ respectively), the DL axiom enforces $(a, c) \in f^{\mathcal{I}}$. The DL axiom does not capture the rule's semantics exactly: open answer sets have to be minimal such that an open answer set cannot contain $f(a, c)$ without applying the body of the rule from R. It seems that the satisfaction of ground rules can indeed

[6] Note that other entities than *team* could have these properties, e.g., a *club* – in the example clubs and teams would then be the same.

be simulated by DL axioms, however, the minimality of open answer sets cannot be captured as such. Note that DL-safe rules are not interpreted by such a minimal model semantics such that it is more likely that they actually could be captured as DL axioms (provided the particular DL allows for nominals). This is subject for further research. Writing non-ground DL-safe rules directly as DL axioms seems to be more intricate, if possible at all.

It is still up to a knowledge engineer to decide whether the minimality property is required to represent the domain under consideration.

5 Related Work

In [22], the language $\mathcal{L}_0$ of a program P is expanded with an infinite sequence of new constants $c_1, \ldots, c_k, \ldots$ such that $\mathcal{L}_k$ is the expansion of $\mathcal{L}_0$ with $c_1, \ldots, c_k$. A pair $\langle k, B\rangle$ for a nonnegative integer k and a set of ground literals B in $\mathcal{L}_k$ is then a k-*belief set* of P iff B is an answer set of P_k, where P_k is the grounding of P in the language $\mathcal{L}_k$. Our definition of open answer sets is more general in the sense that also infinite universes are allowed, while a k-belief set is always finite. Nonetheless, the other direction is valid: every k-belief set can be written as an open answer set.

Defining k-belief sets, or open answer sets for that matter, easily leads to undecidability as was argued for k-belief sets in [48]. Interestingly, [48] shows that reasoning becomes decidable again under the well-founded semantics . Since for stratified programs this semantics coincides with the answer set semantics, one has decidability of reasoning for k-belief sets of stratified programs. However, trying to extend the language of stratified programs with an extra stratum below all others, containing disjunctions of positive literals, leads to undecidability again [48]. Considering, in this light, $\Phi(C, \Sigma)$, which basically consists of a stratified part, defining the DLs constructors, and a disjunctive part, the free rules, we have, however, still decidability, emphasizing the importance of the forest-model property.

Another approach to infinite reasoning, besides infinite open domains, is presented in [11], where function symbols are included in the language. *Finitary programs* are identified as a class for which ground query answering is decidable, and lead to elegant formulations of, e.g., plans with unbounded planning length. Formally, they are defined as programs that are finitely recursive, i.e., every ground atom may only depend on a finite number of other ground atoms, and such that only a finite number of odd-cycles may occur in the grounded program. Neither conditions are necessary for FoLPs: the local FoLP containing rules $a(X) \leftarrow f(X, Y), not\ b(Y)$ and $b(X) \leftarrow a(X)$, when grounded with an infinite universe, is not finitely recursive and contains infinitely many odd-cycles. Since not all finitary programs are FoLPs, both classes of programs are not directly related, and the forest-model property appears to be an alternative indication of “finitary” reasoning with possibly infinite knowledge. While ground query answering with finitary programs is decidable, unground query answering is only semi-decidable [11]. Since both are decidable for FoLPs, FoLPs are arguably more suited for checking consistency of, e.g., ontologies. Moreover, checking whether a program is finitary is itself undecidable, in contrast with FoLPs, which are a syntactic restriction of DLPs.

There are basically two lines of research that try to reconcile description logics with logic programming. The approaches in [10,24,44,2,38,52] simulate DLs with LP, possibly with a detour to FOL, while [15,47,17] attempt to unite the strengths of DLs and LP by letting them coexist and interact.

In [10], 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 opennes lies in the use of undefined predicates, which are comparable to free predicates with the difference that free predicates can be expressed within the FoLP 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 $\mathcal{ALCHOQ}(\sqcup, \sqcap)$.

[24] 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, [24]'s forte lies in the identification of a subclass of DLs that make efficient reasoning through LPs possible. [44] extends the work in [24], 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.

[38] and [52] simulate reasoning in DLs with a LP formalism by using an intermediate translation to first-order clauses. In [38], $\mathcal{SHIQ}^-$ 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. Note that [38] can deal with transitive roles which is a clear advantage over our approach in the context of DL simulation.

[52] translates $\mathcal{ALCQI}$ concepts to first-order formulas, 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 FoLPs can be used both for reasoning with DLs and for a direct and elegant expression of knowledge. Furthermore, FoLPs 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 [15] 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 [47] for disjunctive Datalog and the $\mathcal{ALC}$ DL. A further generalization is attained in [17] where the particular DL can be the expressive $\mathcal{SHOIN}(\mathbf{D})$. The DL knowledge base is considered 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 [45], 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 FoLPs can be reduced to finite ASP, it can be trivially reduced to the approach in [17] with an empty DL knowledge base. In [18] the approach of [17] was adapted for the well-founded semantics instead of the answer set semantics.

In [4], one builds a nonmonotonic rule system on top of the ontology language DAML+OIL [8], a predecessor of OWL. More specifically, they use *defeasible logic* [46] to express rule-based knowledge and argue its use for E-commerce applications on the Semantic Web. Another approach that combines DAML+OIL with rules can be found in [25], where *situated courteous logic programs* in the rule markup language RuleML [1] provide for the nonmonotonic rule system.

A notable approach, which cannot be categorized in one of the two lines of research described above, although it tends towards the coexisting approach, is the SWRL [36] 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 [33] but lacks, nevertheless, interesting knowledge representation mechanisms such as negation as failure.

[23] explains how reasoning with SWRL [36], can be done by iteratively calling the DL reasoner RACER [26] and the rule-based reasoner *Jess* [20], 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. On the other hand, the approach in [23] is quite ad hoc

and not formally proved to be correct. A similar iterative angle is taken in [43] 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 the semantics of answer set programming with support for open domains. This extension led to an increase in expressiveness, but also to undecidability of reasoning. This was remedied by syntactically restricting the types of allowed rules in logic programs, resulting in extended forest logic programs. We further restricted EFoLPs to local EFoLPs that have the bounded finite model property. Lower and upper bounds for the complexity of reasoning were established.

Furthermore, we showed how EFoLPs can simulate reasoning in a DL that is related to the OWL DL ontology language together with DL-safe rules. A disadvantage of the EFoLP approach, however, compared to state-of-the-art DLs, is the inability to express transitive roles as in, e.g., the DL $\mathcal{SHIQ}$: we restrict ourselves to EFoLPs with the local model property in order to ensure a bounded finite model property, a restricting property that $\mathcal{SHIQ}$ does not have.

Since EFoLP is a logic programming paradigm, with, e.g., negation as failure and the consequential nonmonotonic reasoning, we believe that EFoLPs may be useful for reasoning with both rules and ontologies on the Semantic Web, and this in such a way that both types of knowledge are fully integrated. We concluded with a description of related work.

It would be interesting to look for further extensions of the forest-model property of EFoLPs. Other syntactical classes of open answer set programming, e.g., *guarded programs* [30], can be identified, based on other decidability vehicles like, e.g., fixed point logic.

Acknowledgements

Davy Van Nieuwenborgh is supported by the Flemish Fund for Scientific Research (FWO-Vlaanderen).

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.

[4] G. Antoniou. A Nonmonotonic Rule System using Ontologies. CEUR Proceedings, 2002.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.

[6] F. Baader and U. Sattler. Number Restrictions on Complex Roles in Description logics. In *Proc. of KR-96*, pages 328–339, 1996.

[7] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[8] 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.

[9] 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. W3C Recommendation - `http://www.w3.org/TR/owl-ref/`, February 2004.

[10] K. Van Belleghem, M. Denecker, and D. De Schreye. A Strong Correspondence between Description Logics and Open Logic Programming. In *Proc. of ICLP'97*, pages 346–360, 1997.

[11] P. A. Bonatti. Reasoning with Infinite Stable Models. *Artificial Intelligence*, 156:75–111, 2004.

[12] A. Borgida. On the Relative Expressiveness of Description Logics and predicate logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.

[13] 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.

[14] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[15] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *J. of Intelligent and Cooperative Inf. Systems*, 10:227–252, 1998.

[16] T. Eiter, W. Faber, M. Fink, G. Pfeifer, and S. Woltran. Complexity of Model Checking and Bounded Predicate Arities for Non-ground Answer Set Programming. In Didier Dubois, Christopher Welty, and Mary-Anne Williams, editors, *Proceedings Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), June 2-5, Whistler, British Columbia, Canada*, pages 377–387. Morgan Kaufmann, 2004.

[17] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. In *Proc. of KR 2004*.

[18] 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.

[19] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier Science Publishers B.V., 1990.

[20] E.J. Friendman-Hill. Jess homepage. http://herzberg.ca.sandia.gov/jess/.

[21] 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.

[22] M. Gelfond and H. Przymusinska. Reasoning in Open Domains. In *Logic Programming and Non-Monotonic Reasoning*, pages 397–413. MIT Press, 1993.

[23] 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.

[24] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57, 2003.

[25] 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.

[26] V. Haarslev and R. Moller. Description of the RACER System and its Applications. In *Proc. of Description Logics 2001*, 2001.

[27] J. Y. Halpern and Y. Moses. A Guide to Completeness and Complexity for Modal Logics of Knowledge and Belief. *Artif. Intell.*, 54(3):319–379, 1992.

[28] 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.

[29] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Nonmonotonic Ontological and Rule-based Reasoning with Extended Conceptual Logic Programs. In *Proc. of ESWC 2005*, LNCS. Springer, 2005. To Appear.

[30] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Guarded Open Answer Set Programming. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *8th International Conference on Logic Programming and Non Monotonic Reasoning (LPNMR 2005)*, number 3662 in LNAI, pages 92–104, Diamante, Italy, September 2005. Springer.

[31] 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.

[32] J. Hladik and U. Sattler. A Translation of Looping Alternating Automata to Description Logics. In *Proc. of CADE-19*, volume 2741 of *LNAI*. Springer, 2003.

[33] 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.

[34] 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.

[35] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proc. of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705, pages 161–180. Springer-Verlag, 1999.

[36] 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.

[37] Ian Horrocks and Peter Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *J. of Web Semantics*, 2004. To Appear.

[38] 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.

[39] Katsumi Inoue and Chiaki Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78, April 1998.

[40] R. E. Ladner. The Computational Complexity of Provability in Systems of Modal Propositional Logic. *SIAM J. Comput.*, 6(3):467–480, 1977.

[41] N. Leone, G. Pfeifer, and W. Faber. DLV homepage. http://www.dbai.tuwien.ac.at/proj/dlv/.

[42] V. Lifschitz. Answer Set Programming and Plan Generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] R. Rosati. Towards Expressive KR Systems Integrating Datalog and Description Logics: Preliminary Report. In *Proc. of DL'99*, pages 160–164, 1999.

[48] J. Schlipf. Some Remarks on Computability and Open Domain Semantics. In *Proc. of the Workshop on Structural Complexity and Recursion-Theoretic Methods in Logic Programming of the International Logic Programming Symposium*, 1993.

[49] M. Schmidt-Schaub and G. Smolka. Attributive Concept Descriptions with Complements. *Artif. Intell.*, 48(1):1–26, 1991.

[50] P. Simons. SMODELS homepage. http://www.tcs.hut.fi/Software/smodels/.

[51] M. Smith, C. Welty, and D. McGuinness. OWL Web Ontology Language Guide. `http://www.w3.org/TR/owl-guide/`, 2004.

[52] T. Swift. Deduction in Ontologies via Answer Set Programming. In Vladimir Lifschitz and Ilkka Niemelä, editors, *Proc. of LPNMR 2004*, volume 2923 of *LNCS*, pages 275–288. Springer, 2004.

[53] M. Y. Vardi. Why is Modal Logic so Robustly Decidable? Technical Report TR97-274, Rice University, April 12, 1997.

[54] M. Y. Vardi. Reasoning about the Past with Two-Way Automata. In *Proc. of ICALP ’98*, pages 628–641. Springer, 1998.