Hybrid Reasoning with Forest Logic Programs*

Cristina Feier and Stijn Heymans

Knowledge-Based Systems Group, Institute of Information Systems
Vienna University of Technology
Favoritenstrasse 9-11, A-1040 Vienna, Austria
{feier,heymans}@kr.tuwien.ac.at

Abstract. Open Answer Set Programming (OASP) is an attractive framework for integrating ontologies and rules. Although several decidable fragments of OASP have been identified, few reasoning procedures exist. In this paper, we provide a sound, complete, and terminating algorithm for satisfiability checking w.r.t. forest logic programs, a fragment where rules have a tree shape and allow for inequality atoms and constants. We further introduce f-hybrid knowledge bases, a hybrid framework where $\mathcal{SHOQ}$ knowledge bases and forest logic programs co-exist, and we show that reasoning with such knowledge bases can be reduced to reasoning with forest logic programs only. We note that f-hybrid knowledge bases do not require the usual (weakly) DL-safety of the rule component, providing thus a genuine alternative approach to hybrid reasoning.

1 Introduction

Integrating Description Logics (DLs) with rules for the Semantic Web has received considerable attention with approaches such as *Description Logic Programs* [7], *DL-safe rules* [18], $\mathcal{DL}$+*log* [20], *dl-programs* [3], *Description Logic Rules* [14], and Open Answer Set Programming (OASP) [12]. OASP combines attractive features from the DL and the Logic Programming (LP) world: an open domain semantics from the DL side allows for stating generic knowledge, without the need to mention actual constants, and a rule-based syntax from the LP side supports nonmonotonic reasoning via *negation as failure*.

Several decidable fragments of OASP were identified by syntactically restricting the shape of logic programs, while carefully safe-guarding enough expressiveness for integrating rule- and ontology-based knowledge. Notable fragments are *Conceptual Logic Programs (CoLPs)* [9] that are able to simulate reasoning in the DL $\mathcal{SHIQ}$ and *Forest Logic Programs (FoLPs)* [10] that are expressive enough to deal with $\mathcal{SHOQ}$. Note that both $\mathcal{SHOQ}$ and $\mathcal{SHIQ}$ are close family of $\mathcal{SHOIN}(\mathbf{D})$, the DL underlying the Web Ontology language OWL-DL [23]. A serious shortcoming of these decidable fragments is their lack of effective reasoning procedures. In [5], we took a first step in mending this by providing a

* This work is partially supported by the Austrian Science Fund (FWF) under the projects P20305 and P20840, and by the European Commission under the project OntoRule (IST-2009-231875).

sound and complete algorithm for *simple CoLPs*. Simple CoLPs restrict CoLPs by disallowing the use of inverse predicates, inequality and restricting the literal cyclity, but that are still expressive enough to simulate the DL $\mathcal{ALCH}$.

In this paper, we extend the algorithm of [5] to *Forest Logic Programs*, an extension of simple CoLPs with constants and inequality and no cyclicity restriction. FoLPs are able to simulate the DL $\mathcal{SHOQ}$. They serve as well as an underlying integration vehicle for ontologies and rules. In order to illustrate this, we define *f-hybrid knowledge bases (fKBs)*, consisting of a $\mathcal{SHOQ}$ knowledge base and a rule component that is a FoLP, with a nonmonotonic semantics similar to the semantics of *g-hybrid knowledge bases* [8], *r-hybrid knowledge bases* [21], and $\mathcal{DL}$+*log* [20]. Our approach differs in two points with current other proposals: (1) In contrast with Description Logic Programs, DL-safe rules, and Description Logic Rules, fKBs have, in line with traditional logic programming, a minimal model semantics for the rule component, thus allowing for nonmonotonic reasoning. (2) To ensure effective reasoning, we do not rely on a (weakly) DL-safeness condition such as [18, 20, 21], which restricts the interaction of the rule component with the DL component. Instead, we rely on a translation to FoLPs.

The major contributions of the paper can be summarized as follows:

- We define in Section 4 a nondeterministic algorithm for deciding satisfiability w.r.t. FoLPs, inspired by tableaux-based methods from DLs. We show that this algorithm is terminating, sound, complete, and runs in 2-NEXPTIME. The algorithm is non-trivial from two perspectives: both the minimal model semantics of OASP, compared to the model semantics of DLs, as well as the open domain assumption, compared to the closed domain assumption of ASP [6], pose specific challenges.
- We show in Section 5 that FoLPs are expressive enough to simulate the DL $\mathcal{SHOQ}$ with *fKBs*, an alternative characterization for hybrid representation and (nonmonotonic) reasoning of knowledge, that supports a tight integration of ontologies and rules.

Extended examples, explanations, and detailed proofs, can be found in [4].

2 Preliminaries

We recall the open answer set semantics [12]. *Constants* $a, b, c, \ldots$, *variables* $x, y, \ldots$, *terms* $s, t, \ldots$, and *atoms* $p(t_1, \ldots, t_n)$ are as usual. We allow for *equality atoms* $s = t$. A *literal* is an atom L or a negated atom *not* L. An *inequality literal* *not* $(s = t)$ will often be denoted with $s \neq t$. An atom (literal) that is not an equality atom (inequality literal) will be called a *regular atom (literal)*. For a set α of literals or (possibly negated) predicates, $\alpha^+ = \{l \mid l \in \alpha\}$ and $\alpha^- = \{l \mid \mathit{not}\ l \in \alpha\}$. For a set X of atoms, $\mathit{not}\ X = \{\mathit{not}\ l \mid l \in X\}$. For a set of (possibly negated) predicates α, $\alpha(x) = \{a(x) \mid a \in \alpha\}$ and $\alpha(x, y) = \{a(x, y) \mid a \in \alpha\}$.

A *program* is a countable set of rules $\alpha \leftarrow \beta$, where α is a finite set of regular literals and β is a finite set of literals. The set α is the *head* and represents a disjunction, while β is the *body* and represents a conjunction. If $\alpha = \emptyset$, the

rule is called a *constraint*. *Free rules* are rules $q(t_1,\ldots,t_n) \vee not\ q(t_1,\ldots,t_n) \leftarrow$ for terms $t_1,\ldots,t_n$; they enable a choice for the inclusion of atoms. We call a predicate q *free* if there is a $q(x_1,\ldots,x_n) \vee not\ q(x_1,\ldots,x_n) \leftarrow$, with variables $x_1,\ldots,x_n$. Atoms, literals, rules, and programs that do not contain variables are *ground*. For a rule or a program X, let $cts(X)$ be the constants in X, $vars(X)$ its variables, and $preds(X)$ its predicates with $upreds(X)$ the unary and $bpreds(X)$ the binary predicates. A *universe* U for P is a non-empty countable superset of the constants in P: $cts(P) \subseteq U$. We call P_U the ground program obtained from P by substituting every variable in P by every element in U. Let $\mathcal{B}_P$ ($\mathcal{L}_P$) be the set of regular atoms (literals) that can be formed from a ground program P.

An *interpretation* I of a ground P is a subset of $\mathcal{B}_P$. We write $I \models p(t_1,\ldots,t_n)$ if $p(t_1,\ldots,t_n) \in I$ and $I \models not\ p(t_1,\ldots,t_n)$ if $I \not\models p(t_1,\ldots,t_n)$. Also, for ground terms s,t, we write $I \models s = t$ if $s = t$ and $I \models not\ s = t$ or $I \models s \neq t$ if $s \neq t$. For a set of ground literals X, $I \models X$ if $I \models l$ for every $l \in X$. 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 $I \models \beta$. A ground constraint $\leftarrow \beta$ is satisfied w.r.t. I if $I \not\models \beta$.

For a ground program P without *not*, an interpretation I of P is a *model* of P if I satisfies every rule in P; it is an *answer set* of P if it is a subset minimal model of P. For ground programs P containing *not*, the *GL-reduct* [6] w.r.t. I is defined as P^I, where P^I contains $\alpha^+ \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in P, $I \models not\ \beta^-$ and $I \models \alpha^-$. I is an *answer set* of a ground P if I is an answer set of P^I.

Below, a program is assumed to be a finite set of rules; infinite programs only appear as byproducts of grounding with an infinite universe. An *open interpretation* of a program P is a pair (U, M) where U is a universe for P and M is an interpretation of P_U. An *open answer set* of P is an open interpretation (U, M) of P with M an answer set of P_U. An n-ary predicate p in P is *satisfiable* if there is an open answer set (U, M) of P s. t. $p(x_1,\ldots,x_n) \in M$.

We introduce notation for trees which extend those in [25]. Let $\cdot$ be a concatenation operator between sequences of constants or natural numbers. A *tree* T with root c (T_c), where c is a specially designated constant, has as nodes sequences of the form $c \cdot s$, where s is a (possibly empty) sequence of positive integers formed with the concatenation operator; for $x \cdot d \in T$, $d \in \mathbb{N}^*$, we have that $x \in T$. For $x \in T$, we call $succ_T(x) = \{x \cdot n \in T \mid n \in \mathbb{N}^*\}$[1], *successors* of x. The set $A_T = \{(x, y) \mid x, y \in T, \exists n \in \mathbb{N}^* : y = x \cdot n\}$ is the set of arcs of a tree T. For $x, y \in T$, we say that $x <_T y$ iff x is a prefix of y and $x \neq y$.

A *forest* F is a set of trees $\{T_c \mid c \in C\}$, where C is a set of distinguished constants. $N_F = \cup_{T \in F} T$ and $A_F = \cup_{T \in F} A_T$ are the set of nodes and the set of arcs of a forest F. For $x \in F$, its successors in F are $succ_F(x) = succ_T(x)$ for $x \in T$. Let $<_F$ be a strict partial order relationship on the set of nodes N_F of a forest F where $x <_F y$ iff $x <_T y$ for some tree T in F. An extended forest EF is a tuple (F, ES) where $F = \{T_c \mid c \in C\}$ is a forest and $ES \subseteq N_F \times C$. ES extends the successor relation: $succ_{EF}(x) = \{y \mid y \in succ_F(x) \text{ or } (x, y) \in ES\}$. A path in EF is a sequence of nodes linked by the successor relation. We denote by $N_{EF} = N_F$ the nodes of EF and by $A_{EF} = A_F \cup ES$ its arcs.

[1] $\mathbb{N}^*$ is the set of positive integers

Finally, for a directed graph G, $paths_G$ is the set of pairs of nodes for which there exists a path in G from the first node in the pair to the second one.

3 Forest Logic Programs

Forest Logic Programs (FoLPs) were introduced in [10] as a syntactical fragment of OASP. FoLPs are a generalization of Conceptual Logic Programs [9] which are logic programs with tree-shaped rules for which satisfiability checking under the open answer set semantics is decidable. FoLPs impose the same structure for rules as CoLPs, but also allow for constants.

Definition 1. *A* forest logic program (FoLP) *is a program with only unary and binary predicates, and such that a rule is either a* free rule $a(s) \vee not\ a(s) \leftarrow$ *or* $f(s,t) \vee not\ f(s,t) \leftarrow$ *, where* s *and* t *are terms such that if* s *and* t *are both variables, they are different* [2]*, a unary rule*

$$r : a(s) \leftarrow \beta(s), (\gamma_m(s,t_m), \delta_m(t_m))_{1 \leq m \leq k}, \psi \tag{1}$$

where s *and* t_m*,* $1 \leq m \leq k$*, are terms (again, if both* s *and* t_m *are variables, they are different; similarly for* t_i *and* t_j*), where*

1. $\psi \subseteq \bigcup_{1 \leq i \neq j \leq k} \{t_i \neq t_j\}$ *and* $\{\neq\} \cap \gamma_m = \emptyset$ *for* $1 \leq m \leq k$*,*
2. $\forall t_i \in vars(r) : \gamma_i^+ \neq \emptyset$*, i.e., for variables* t_i *there is a positive atom that connects* s *and* t_i*,*

or a binary rule

$$f(s,t) \leftarrow \beta(s), \gamma(s,t), \delta(t) \tag{2}$$

with $\{\neq\} \cap \gamma = \emptyset$ *and* $\gamma^+ \neq \emptyset$ *if* t *is a variable (*s *and* t *are different if both are variables), or a constraint* $\leftarrow a(s)$ *or* $\leftarrow f(s,t)$ *where* s *and* t *are different if both are variables).*

Constraints can be left out without losing expressiveness: $\leftarrow$ *body* can be replaced by a rule of the form $co(x) \leftarrow not\ co(x), body$, for a new predicate co. As their name suggests FoLPs have the *forest model property*:

Definition 2. *Let* P *be a program. A predicate* $p \in upreds(P)$ *is* forest satisfiable *w.r.t.* P *if there is an open answer set* (U, M) *of* P *and there is an extended forest* $EF \equiv (\{T_\varepsilon\} \cup \{T_a \mid a \in cts(P)\}, ES)$*, where* ε *is a constant, possibly one of the constants appearing in* P [3]*, and a labeling function* $\mathcal{L} : \{T_\varepsilon\} \cup \{T_a \mid a \in cts(P)\} \cup A_{EF} \rightarrow 2^{preds(P)}$ *s. t.*

- $U = N_{EF}$*, and*
- $p \in \mathcal{L}(\varepsilon)$*,*

[2] A rule $f(X,X) \vee not\ f(X,X) \leftarrow$ is not allowed.

[3] Note that in this case $T_\varepsilon \in \{T_a \mid a \in cts(P)\}$. Thus, the extended forest contains for every constant from P a tree which has as root that specific constant and possibly, but not necessarily, an extra tree with unidentified root node.

– $z \cdot i \in T \in EF$, $i > 0$, *iff there is some* $f(z, z \cdot i) \in M$, $z \in T_x$, *and*
– *for* $y \in T \in EF$, $q \in upreds(P)$, $f \in bpreds(P)$, *we have that*
  • $q(y) \in M$ *iff* $q \in \mathcal{L}(y)$, *and*
  • $f(y, u) \in M$ *iff* $(u = y \cdot i \vee u \in cts(P)) \wedge f \in \mathcal{L}(y, u)$.

We call such a (U, M) *a* forest model *and a program* 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.

Proposition 1. *FoLPs have the forest model property.*

In [5], we introduced the class of simple Conceptual Logic Programs. It is easy to see that every simple CoLP is an FoLP. As satisfiability checking w.r.t. simple Conceptual Logic Programs is EXPTIME-hard, the following property follows:

Proposition 2. *Satisfiability checking w.r.t. FoLPs is* EXPTIME*-hard.*

4 An Algorithm for Forest Logic Programs

In this section, we define a sound, complete, and terminating algorithm for satisfiability checking w.r.t. FoLPs. In [11] it has been shown that several restrictions of FoLPs which have the finite model property are decidable, but there was no result so far regarding the whole fragment. Thus, the algorithm described in this section also establishes a decidability result for FoLPs. The algorithm also relies on the finite model property of FoLPs the proof of which is one of the main steps of the completeness proof.

For every non-free predicate q and a FoLP P, let P_q be the rules of P that have q as a head predicate. For a predicate p, $\pm p$ denotes p or *not* p, whereby multiple occurrences of $\pm p$ in the same context will refer to the same symbol (either p or *not* p). The negation of $\pm p$ (in a given context) is $\mp p$, that is, $\mp p =$ *not* p if $\pm p = p$ and $\mp p = p$ if $\pm p =$ *not* p.

The basic data structure for our algorithm is a *completion structure*.

Definition 3. *A completion structure for a FoLP* P *is a tuple* $\langle EF, G,$ CT, ST, $bl\rangle$ *where* $EF = (F, ES)$ *is an extended forest which together with the labeling functions* CT *and* ST *and the set* bl *of blocking nodes is used to represent/construct a tentative forest model.* $G = \langle N, A\rangle$ *is a directed graph with nodes* $N \subseteq \mathcal{B}_{P_{N_{EF}}}$ *and edges* $A \subseteq N \times N$ *which is used to keep track of dependencies between elements of the constructed model (the positive atom dependency graph of* $P_{N_{EF}}$*): N represents the tentative model, while* N_{EF} *represents the tentative universe.*

The content *function* $\text{CT} : N_{EF} \cup A_{EF} \rightarrow 2^{preds(P) \cup not\ (preds(P))}$ *maps a node to a set of (possibly negated) unary predicates and an arc to a set of (possibly negated) binary predicates such that* $\text{CT}(x) \in 2^{upreds(P) \cup not(upreds(P))}$ *if* $x \in N_{EF}$, *and* $\text{CT}(x) \in 2^{bpreds(P) \cup not(bpreds(P))}$ *if* $x \in A_{EF}$. *Every presence of a non-negated predicate symbol* p *in the content of some node/arc* x *of* EF *indicates that* $p(x)$ *is part of the tentative model represented by* EF.

The status *function* $\text{ST} : \{(x,q) \mid q \in \text{CT}(x), x \in N_{EF} \cup A_{EF}\} \cup \{(x,q) \mid not\ q \in \text{CT}(x), x \in A_{EF}\} \cup \{(x, not\ q, r) \mid not\ q \in \text{CT}(x), x \in N_{EF}, r \in P_q\} \rightarrow \{exp, unexp\}$ *attaches to every (possibly negated) predicate which appears in the content of a node/arc* x *of* EF *a status value which indicates whether the predicate has already been expanded in that node/edge. As it will be indicated later, the completion structure is evolved such that the presence/absence of any predicate symbol in the content of some node/arc is justified, so it is necessary to keep track which predicate symbols have already been justified in every node/arc of* EF*. As negative unary predicates have to be justified by showing that no rule which defines them can be applied and this can be done in several steps, the function takes as an argument also a rule for such negated predicate symbols.*

An initial completion structure *for checking satisfiability of a unary predicate* p *w.r.t. a FoLP* P *is a completion structure* $\langle EF, G, \text{CT}, \text{ST}, bl\rangle$ *with* $EF = (F, ES)$*,* $F = \{T_\varepsilon\} \cup \{T_a \mid a \in cts(P)\}$*, where* ε *is a constant, possibly in* $cts(P)$*,* $T_x = \{x\}$*, for* $x \in \{\varepsilon\} \cup cts(P)$*,* $ES = \emptyset$*,* $G = \langle V, E\rangle$*,* $V = \{p(\varepsilon)\}$*,* $E = \emptyset$*, and* $\text{CT}(\varepsilon) = \{p\}$*,* $\text{ST}(\varepsilon, p) = unexp$*,* $bl = \emptyset$*.*

The forest is initialized with single-node trees with roots constants from P and, possibly, a new single-node tree with an anonymous root. The anonymous root, in case it exists, contains p, the predicate to satisfy. Otherwise the root of one of the other trees contains p. G is a graph with a single vertex $p(\varepsilon)$.

Next, we will show how to evolve an initial completion structure for checking the satisfiability of a unary predicate p w.r.t. a FoLP P by means of *expansion rules* to a complete clash-free structure that corresponds to a finite representation of an open answer set in case p is satisfiable w.r.t. P. *Applicability rules* state the necessary conditions to apply these expansion rules.

4.1 Expansion Rules

The expansion rules update the completion structure whenever justifying a literal l in the current model leads to the presence of new literal $\pm p(z)$ in the model. Thus, $\pm p$ is inserted in the content of z if it is not already there and marked as unexpanded (w.r.t. all rules which define p in case it is a negative instance and z is a node in EF). Also, in case $\pm p(z)$ is an atom, it is ensured that it is a node in G; moreover, if l is also an atom, a new arc from l to $\pm p(z)$ is created to indicate the dependency between l and $\pm p(z)$ in the model. Formally:

- if $\pm p \notin \text{CT}(z)$, then $\text{CT}(z) = \text{CT}(z) \cup \{\pm p\}$ and $\text{ST}(z, \pm p) = unexp$ in case $\pm p = p$ or $b \in bpreds(()P)$ and $\text{ST}(z, \pm p, r) = unexp$, for all $r \in P_p$ in case $\pm p = not\ p$ and $p \in upreds(()P)$,
- if $\pm p = p$ and $\pm p(z) \notin V$, then $V = V \cup \{\pm p(x)\}$,
- if $l \in \mathcal{B}_{P_{N_{EF}}}$ and $\pm p = p$, then $E = E \cup \{(l, \pm p(z))\}$.

As a shorthand, we denote this sequence of operations as $update(l, \pm p, z)$; more general, $update(l, \beta, z)$ for a set of (possibly negated) predicates β, denotes $\forall \pm a \in \beta, update(l, \pm a, z)$.

In the following, for a completion structure $\langle EF, G, \text{CT}, \text{ST}, bl\rangle$, let $x \in N_{EF}$ and $(x, y) \in A_{EF}$ be the node, resp. arc, under consideration.
(i) Expand unary positive. For a unary positive (non-free) $p \in \text{CT}(x)$ s. t. $\text{ST}(x, p) = \mathit{unexp}$,

- nondeterministically choose a rule $r \in P_p$ of the form (1) s. t. s unifies with x. The rule will motivate $p(x)$ in the tentative open answer set.
- for the β in the body of r, $\mathit{update}(p(x), \beta, x)$,
- pick up (or define when needed) k successors for x, $(y_m)_{1\leq m\leq k}$, s. t.:
  - for every $1 \leq i, j \leq k$ s. t. $t_i \neq t_j \in \psi$: $y_i \neq y_j$;
  - for every $1 \leq m \leq k$: $y_m \in \mathit{succ}_{EF}(x)$ or y_m is a new EF-successor of x:
    - $y_m = x \cdot n$, $n \in \mathbb{N}^*$ and $x \cdot n \notin \mathit{succ}_{EF}(x)$. Update T_c, where $x \in T_c$: $T_c = T_c \cup \{y_m\}$, or
    - $y_m = a$, where $a \in \mathit{cts}(P)$ and $a \notin \mathit{succ}_{EF}(x)$; $ES = ES \cup (x, a)$.
- for every successor y_m of x, $1 \leq m \leq k$: $\mathit{update}(p(x), \gamma_m, (x, y_m))$ and $\mathit{update}(p(x), \delta_m, y_m)$.
- set $\text{ST}(x, p) = \mathit{exp}$.

(ii) Expand unary negative. Justifying a negative unary predicate $\mathit{not}\ p \in \text{CT}(x)$ (the absence of $p(x)$ in the constructed model) means refuting the body of every ground rule which defines $p(x)$. The body of such a rule is refuted either

- locally: a literal from $\beta(x)$ has to be refuted which amounts to a certain $\pm q \in \beta$ not appearing in $\text{CT}(x)$, or
- depending on the particular grounding of the rule, in an outgoing arc of x, or in a successor of x: thus, some $\pm f \in \delta_m$ should not appear in $\text{CT}(x, y_j)$ or some $\pm q \in \gamma_m$ should not appear in $\text{CT}(y_j)$, where $y_j \in \mathit{succ}_F(x)$, or
- if there is no valid assignment of successors of x to successors of s in the rule which fulfill the inequalities in the rule.

Formally, for a unary negative (non-free) $\mathit{not}\ p \in \text{CT}(x)$ and a rule $r \in P_p$ of the form (1) s. t. x unifies with s (s is the term from the head of the rule) and $\text{ST}(x, \mathit{not}\ p, r) = \mathit{unexp}$ do one of:

- choose a $\pm q \in \beta$, $\mathit{update}(\mathit{not}\ p(x), \mp q, x)$, and set $\text{ST}(x, \mathit{not}\ p, r) = \mathit{exp}$, or
- if for all $p \in \mathit{upreds}(P)$, $p \in \text{CT}(x)$ or $\mathit{not}\ p \in \text{CT}(x)$, and for all $p \in \text{CT}(x)$, $\text{ST}(p, x) = \mathit{exp}$, then for all $y_{i_1}, \ldots, y_{i_k}$ s. t. $(1 \leq i_j \leq n)_{1\leq j\leq k}$, where $\mathit{succ}_{EF}(x) = \{y_1, \ldots y_n\}$: if for all $1 \leq j, l \leq k$, $t_j \neq t_l \in \psi \Rightarrow y_{i_j} \neq y_{i_l}$, then do one of the following:
  - for some m, $1 \leq m \leq k$, pick up a binary (possibly negated) predicate symbol $\pm f$ from δ_m and $\mathit{update}(\mathit{not}\ p(x), \mp f, (x, y_{i_m}))$, or
  - for some m, $1 \leq m \leq k$, pick up a unary negated predicate symbol $\mathit{not}\ q$ from γ_m and $\mathit{update}(\mathit{not}\ p(x), q, y_{i_m})$.

  Set $\text{ST}(x, \mathit{not}\ p, r) = \mathit{exp}$.

One can see that once the body of a ground version of a unary rule $r \in P_p$, for which the head term s is substituted with the current node x, is locally refuted, the bodies of all ground versions of this rule, for which s is substituted with x are

locally refuted, too. For the other two refutation cases all possible groundings of a rule have to be considered and this is not possible until all the successors of x are known. This is the case when all positive predicates in the content of the current node have been expanded and no positive predicate will be further inserted in $\text{CT}(x)$:then, an iteration over all possible groundings of the rule r is triggered. For every grounding if the inequality constraints are violated the body of that particular instantiation of the rule is already refuted; otherwise, one of the body literals from the non-local part of the rule (γs or δs) has to be refuted.
(iii) Choose a unary predicate. If for all $a \in \text{CT}(x)$, $\text{ST}(x, a) = \mathit{exp}$ or a is free, and for all $(x, y) \in A_{EF}$, and for all $\pm f \in \text{CT}(x, y)$, $\text{ST}((x, y), \pm f) = \mathit{exp}$ or f is free, and there is a $p \in \mathit{upreds}(P)$ s. t. $p \notin \text{CT}(x)$ and $\mathit{not}\ p \notin \text{CT}(x)$, then add p to $\text{CT}(x)$ with $\text{ST}(x, p) = \mathit{unexp}$ or add $\mathit{not}\ p$ to $\text{CT}(x)$ with $\text{ST}(x, \mathit{not}\ p, r) = \mathit{unexp}$, for every rule $r \in P_p$.

In other words, if there is a node x for which all positive predicates in its content and all predicates in the contents of its outgoing arcs are free or have already been expanded and if there are still unary predicates which do not appear in $\text{CT}(x)$, one has to pick such a unary predicate symbol p and inject either p or $\mathit{not}\ p$ in $\text{CT}(x)$. This is needed for consistency: it does not suffice to find a justification for the predicate to satisfy, but one also has to show that this justification is part of an actual open answer set, which is done by effectively constructing it. We do not impose that all negative predicate symbols are expanded as that would constrain all the ensuing literals to be locally refuted.

Similarly to rules (i), (ii), and (iii) we define the expansion rules for binary predicates: (iv) *Expand binary positive*, (v) *Expand binary negative*, and (vi) *Choose binary.*

4.2 Applicability Rules

The applicability rules restrict the use of the expansion rules.
(vii) Saturation. A node $x \in N_{EF}$ is *saturated* if for all $p \in \mathit{upreds}(P)$, $p \in \text{CT}(x)$ or $\mathit{not}\ p \in \text{CT}(x)$, and no $\pm q \in \text{CT}(x)$ can be expanded with rules (i-iii), and for all $(x, y) \in A_{EF}$ and $p \in \mathit{bpreds}(P)$, $p \in \text{CT}(x, y)$ or $\mathit{not}\ p \in \text{CT}(x, y)$, and no $\pm f \in \text{CT}(x, y)$ can be expanded with (iv-vi). No expansions should be performed on a node from N_{EF} which does not belong to $\mathit{cts}(P)$ until its predecessor is saturated (constants can have more than one predecessor in the completion, including themselves).
(viii) Blocking. A node $x \in N_{EF}$ is *blocked* if there is an ancestor y of x in F, $y <_F x$, $y \notin \mathit{cts}(P)$, s.t. $\text{CT}(x) \subseteq \text{CT}(y)$ and the set $\mathit{paths}_G(x, y) = \{(p, q) \mid (p(x), q(y)) \in \mathit{paths}_G\}$ is empty. We call (y, x) a *blocking pair* and update bl: $bl = bl \cup \{(y, x)\}$. No expansions can be performed on a blocked node. Intuitively, if there is an ancestor y of x which is not a constant, whose content includes the content of x, and there are no paths in G from a positive literal $p(x)$ to another positive literal $q(y)$ one could reuse the justification for x when dealing with y.
(ix) Redundancy. A node $x \in N_{EF}$ is *redundant* if it is saturated and there are k ancestors of x in F, $(y_i)_{1 \leq i \leq k}$, where $k = (2^{p^2} - 1)(2^p)$, and $p = |\mathit{upreds}(P)|$,

s. t. $\text{CT}(x) = \text{CT}(y_i)$ and (y_i, x) is not a blocking pair for every $1 \leq i \leq n$. In other words, a node is redundant if there are other k nodes on the same branch with the current node which all have content equal to the content of the current node. The presence of a redundant node stops the expansion process. In the completeness proof we show that any forest model of a FoLP P which satisfies p can be reduced to another forest model which satisfies p and has at most $k+1$ nodes with equal content on any branch of a tree from the forest model, and furthermore the $(k+1)st$ node, in case it exists, is blocked. One can thus search for forest models only of the latter type. This rule exploits that result: we discard models which are not in this shrunk search space.

.

4.3 Termination, Soundness, and Completeness

A completion structure is *contradictory* if for some $x \in N_{EF}$ and $a \in upreds(P)$, $\{a, not\ a\} \subseteq \text{CT}(x)$ or for some $(x, y) \in A_{EF}$ and $f \in bpreds(P)$, $\{f, not\ f\} \subseteq \text{CT}(x, y)$. A *complete completion structure* for a FoLP P and a $p \in upreds(P)$, is a completion structure that results from applying the expansion rules to the initial completion structure for p and P, taking into account the applicability rules, s. t. no expansion rules can be further applied. Also, a complete completion structure $CS = \langle EF, G, \text{CT}, \text{ST}, bl \rangle$ is *clash-free* if: (1) CS is not contradictory (2) EF does not contain redundant nodes (3) G does not contain cycles.

We show that an initial completion structure for a unary predicate p and a FoLP P can always be expanded to a complete completion structure (*termination*), that, if p is satisfiable w.r.t. P, there is a clash-free complete completion structure (*soundness*), and, finally, that, if there is a clash-free complete completion structure, p is satisfiable w.r.t. P (*completeness*).

Proposition 3 (termination). *Let P be FoLP and $p \in upreds(P)$. Then, one can construct a finite complete completion structure by a finite number of applications of the expansion rules to the initial completion structure for p w.r.t. P, taking into account the applicability rules.*

Proof Sketch. One cannot introduce infinitely many successors to a node, as the arity of the trees in the completion structure is bound by the number of unary predicates in P and the number of variables in unary rules. Also, every infinite path in the extended forest eventually contains $|k+1|$ saturated nodes with equal content, where k is as in the redundancy rule, and thus either a blocked or a redundant node, which is not further expanded. □

Proposition 4 (soundness). *Let P be a FoLP and $p \in upreds(P)$. If there is a clash-free complete completion structure for p w.r.t. P, then p is satisfiable w.r.t. P.*

Proposition 5 (completeness). *Let P be a FoLP and $p \in upreds(P)$. If p is satisfiable w.r.t. P, then there exists a clash-free complete completion structure for p w.r.t. P.*

4.4 Complexity Results

Let $CS = \langle EF, G, \text{CT}, \text{ST}, bl \rangle$ be a complete completion structure. Every path of a tree in EF contains at most $k+1$ nodes with equal content (as suggested by the applicability rules (viii) and (ix)), where k is as defined in the redundancy rule; thus, there are at most $(k+1)2^n$ nodes on every such path, where $n = |upreds(P)|$. The branching of every tree is bound by a constant q which is a linear function of the number of variables in unary rules from P. Thus, there are at most $q^{(k+1)2^n}$ nodes in a tree, and at most $(c+1)q^{(k+1)2^n}$ nodes in EF, where c is the number of constants present in the program at hand. So the amount of nodes in CS is double exponential in the size of P, and the algorithm runs in 2-NEXPTIME.

Note that such a high complexity is expected when dealing with tableaux-like algorithms. For example in Description Logics, although satisfiability checking in $\mathcal{SHIQ}$ is EXPTIME-complete, practical algorithms run in 2-NEXPTIME [24].

5 F-hybrid Knowledge Bases

In this section, we introduce *f-hybrid* knowledge bases (fKBs), a formalism that combines KBs expressed in the Description Logic $\mathcal{SHOQ}$ with FoLPs. We assume the reader is familiar with $\mathcal{SHOQ}$; details can be found in [4].

Note that we assume the *unique name assumption* by imposing that $o^{\mathcal{I}} = o$ for individuals $o \in \mathbf{I}$. Individuals are thus assumed to be part of any domain $\Delta^{\mathcal{I}}$. OWL does not have the unique name assumption [23], and thus different individuals can point to the same resource. However, the open answer set semantics gives a Herbrand interpretation to constants and for consistency we assume that also DL nominals are interpreted this way.

Example 1. Consider the $\mathcal{SHOQ}$ KB Σ with $Father \sqsubseteq \exists child.Human \sqcap \neg Female$ and $\{john\} \sqsubseteq (\leq 2child.Human)$ as axioms. Intuitively, the first terminological axiom says that fathers have a human child and are not female. The second axiom says that *john* has not more than 2 human children.

Definition 4. *An* f-hybrid knowledge base (fKB) *is a pair* $\langle \Sigma, P \rangle$ *where* Σ *is a* $\mathcal{SHOQ}$ *KB and* P *is a FoLP.*

The predicate symbols in P can be concept or role names from Σ, in which case the corresponding atoms/literals are called *DL atoms/literals*, but w.l.o.g. we assume they do not coincide with complex concept or role descriptions. Note that we do not impose Datalog safeness or *(weakly) DL safeness* [19, 21, 20] for the rule component. Intuitively, the restricted shape of FoLPs suffices to guarantee decidability; FoLPs are in general neither Datalog safe nor weakly DL-safe; we discuss the relation with weakly DL-safeness in detail in Section 6.

Example 2. An fKB $\langle \Sigma, P \rangle$, with Σ as in Ex. 1 and P the FoLP $unhappy(X) \leftarrow not\ Father(X)$ indicates that persons that are not fathers are unhappy, where $Father(X)$ is a DL literal.

Similarly as in [8], we define, given a DL interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and a ground program P, the *projection* $\Pi(P,\mathcal{I})$ of P with respect to $\mathcal{I}$, as follows: for every rule r in P,

(a) if there exists a DL literal in the head of the form $A(t_1,\ldots,t_n)$ with $(t_1,\ldots,t_n) \in A^{\mathcal{I}}$, or *not* $A(t_1,\ldots,t_n)$ with $(t_1,\ldots,t_n) \notin A^{\mathcal{I}}$, then delete r,

(b) if there exists a DL literal in the body of the form $A(t_1,\ldots,t_n)$ with $(t_1,\ldots,t_n) \notin A^{\mathcal{I}}$, or *not* $A(t_1,\ldots,t_n)$ with $(t_1,\ldots,t_n) \in A^{\mathcal{I}}$, then delete r, (c) otherwise, delete all DL literals from r. Intuitively, the projection "evaluates" the program with respect to $\mathcal{I}$ by removing (evaluating) rules and DL literals consistently with $\mathcal{I}$; conceptually this is similar to the reduct, which removes rules and negative literals consistently with an interpretation of the program.

Definition 5. *Let $\langle \Sigma, P\rangle$ be an fKB. An* interpretation *of $\langle \Sigma, P\rangle$ is a tuple $(U,\mathcal{I},M)$ s. t. U is a universe for P, $\mathcal{I} = (U,\cdot^{\mathcal{I}})$ is an interpretation of Σ, and M is an interpretation of $\Pi(P_U,\mathcal{I})$. Then, $(U,\mathcal{I},M)$ is a* model *of an fKB $\langle \Sigma, P\rangle$ if $\mathcal{I}$ is a model of Σ and M is an answer set of $\Pi(P_U,\mathcal{I})$.*

The semantics of a fKB $\langle \Sigma, P\rangle$ is s. t. if $\Sigma = \emptyset$, a model of $\langle \Sigma, P\rangle$ corresponds to an open answer set of P, and if $P = \emptyset$, a model of $\langle \Sigma, P\rangle$ corresponds to a DL model of Σ. In this way, the semantics of fKBs is nicely layered on top of both the DL semantics and the open answer set semantics.

Example 3. For the fKB $\langle \Sigma, P\rangle$ in Example 2, take a universe $U = \{john, x\}$ and $\cdot^{\mathcal{I}}$ s. t. $Father^{\mathcal{I}} = \{x\}$, $child^{\mathcal{I}} = \{(x, john)\}$, $Female^{\mathcal{I}} = \emptyset$, $Human^{\mathcal{I}} = U$, and $john^{\mathcal{I}} = john$. It is easy to see that $\mathcal{I} = (U,\cdot^{\mathcal{I}})$ is indeed a model of Σ. With $M = \{unhappy(john)\}$, one can check that $(U,\mathcal{I},M)$ is a model of $\langle \Sigma, P\rangle$.

For p a concept expression from Σ or a predicate from P, we say that p is *satisfiable* w.r.t. (Σ, P) if there is a model $(U,\mathcal{I},M)$ s. t. $p^{\mathcal{I}} \neq \emptyset$ or $p(x_1,\ldots,x_n) \in M$ for some $x_1,\ldots,x_n$ in U, resp.

We can reduce satisfiability checking w.r.t. fKBs to satisfiability checking of FoLPs only. Roughly, for each concept expression one introduces a new predicate together with rules that define the semantics of the corresponding DL construct. Constraints then encode the axioms, and the first-order interpretation of DL concept expressions is simulated using free rules.

For the formal translation, we define the *closure* $clos(\Sigma)$ of a $\mathcal{SHOQ}$ KB Σ as the smallest set satisfying the following conditions:

- for each $C \sqsubseteq D$ an axiom in Σ (role or terminological), $\{C, D\} \subseteq clos(\Sigma)$,
- for each $\mathsf{Trans}(R)$ in Σ, $\{R\} \subseteq clos(\Sigma)$,
- for every D in $clos(\Sigma)$, we have
  - if $D = \neg D_1$, then $\{D_1\} \subseteq clos(\Sigma)$,
  - if $D = D_1 \sqcup D_2$, then $\{D_1, D_2\} \subseteq clos(\Sigma)$,
  - if $D = D_1 \sqcap D_2$, then $\{D_1, D_2\} \subseteq clos(\Sigma)$,
  - if $D = \exists R.D_1$, then $\{R, D_1\} \cup \{\exists S.D_1 \mid S \sqsubseteq R, S \neq R, \mathsf{Trans}(S) \in \Sigma\} \subseteq clos(\Sigma)$,
  - if $D = \forall R.D_1$, then $\{\exists R.\neg D_1\} \subseteq clos(\Sigma)$,

- if $D = (\leq n\ Q.D_1)$, then $\{(\geq n+1\ Q.D_1)\} \subseteq clos(\Sigma)$,
- if $D = (\geq n\ Q.D_1)$, then $\{Q, D_1\} \subseteq clos(\Sigma)$.

Concerning the addition of the extra $\exists S.D_1$ for $\exists R.D_1$ in the closure, note that $x \in (\exists R.D_1)^{\mathcal{I}}$ holds if there is some $(x,y) \in R^{\mathcal{I}}$ with $y \in D_1^{\mathcal{I}}$, i.e., there is some $S \sqsubseteq^* R$ with S transitive (or $S = R$) s. t. $(x,u_0) \in S^{\mathcal{I}}, \ldots, (u_n,y) \in S^{\mathcal{I}}$ with $y \in D_1^{\mathcal{I}}$. This amounts to $x \in (\exists S.D_1)^{\mathcal{I}}$. Thus, in the open answer set setting, we have that $\exists R.D_1(x)$ is in the open answer set if $R(x,y)$ and $D_1(y)$ hold or $\exists S.D_1(x)$ holds for some transitive subrole S of R. The predicate $\exists S.D_1$ will be defined by adding recursive rules, hence the inclusion of such predicates in the closure. Furthermore, for a $(\leq n\ Q.D_1)$ in the closure, we add $\{(\geq n+1\ Q.D_1)\}$, since we will base our definition of the former predicate on the DL equivalence $(\leq n\ Q.D_1) \equiv \neg(\geq n+1\ Q.D_1)$.

We define the FoLP $\Phi(\Sigma)$, obtained from the $\mathcal{SHOQ}$ KB Σ, as follows:

- For each terminological axiom $C \sqsubseteq D \in \Sigma$, add $\leftarrow C(X), not\ D(X)$.
- For each role axiom $R \sqsubseteq S \in \Sigma$, add $\leftarrow R(X, Y), not\ S(X, Y)$.
- Next, we distinguish between the types of concept expressions that appear in $clos(\Sigma)$. For each $D \in clos(\Sigma)$:
  - if D is a concept name, add $D(X) \vee not\ D(X) \leftarrow$,
  - if D is a role name, add $D(X, Y) \vee not\ D(X, Y) \leftarrow$,
  - if $D = \{o\}$, add $D(o) \leftarrow$,
  - if $D = \neg E$, add $D(X) \leftarrow not\ E(X)$,
  - if $D = E \sqcap F$, add $D(X) \leftarrow E(X), F(X)$,
  - if $D = E \sqcup F$, add $D(X) \leftarrow E(X)$ and $D(X) \leftarrow F(X)$,
  - if $D = \exists Q.E$, add $D(X) \leftarrow Q(X, Y), E(Y)$ (5), and for all $S \sqsubseteq^* Q$, $S \neq R$, with $\mathsf{Trans}(S) \in \Sigma$, add rules $D(X) \leftarrow (\exists S.E)(X)$ (6). If $\mathsf{Trans}(Q) \in \Sigma$, we further add the rule $D(X) \leftarrow Q(X, Y), D(Y)$ (7),
  - if $D = \forall R.E$, add $D(X) \leftarrow not\ (\exists R.\neg E)(X)$
  - if $D = (\leq n\ Q.E)$, add $D(X) \leftarrow not\ (\geq n+1\ Q.E)(X)$,
  - if $D = (\geq n\ Q.E)$, add $D(X) \leftarrow Q(X, Y_1), \ldots, Q(X, Y_n), E(Y_1), \ldots, E(Y_n), (Y_i \neq Y_j)_{1 \leq i \neq j \leq n}$

Rule (5) is what one would intuitively expect for the exists restriction. However, in case Q is transitive this rule is not enough. Indeed, if $Q(x,y)$, $Q(y,z)$, $E(z)$ are in an open answer set, one expects $(\exists Q.E)(x)$ to be in it as well if Q is transitive. However, we have no rules enforcing $Q(x,z)$ to be in the open answer set without violating the FoLP restrictions. We can solve this by adding to (5) rule (6) s. t. the chain $Q(x,y)$, $Q(y,z)$, with $E(z)$ in the open answer set correctly deduces $D(x)$. It may still be that there are transitive subroles of Q that need the same recursive treatment as above. To this end, we introduce rule (7). We do not need such a trick with the number restrictions since the roles Q in a number restriction are required to be simple, i.e., without transitive subroles.

Proposition 6. *Let $\langle \Sigma, P \rangle$ be a $\mathcal{SHOQ}$ KB. Then, $\Phi(\Sigma) \cup P$ is a FoLP, and has a size that is polynomial in the size of Σ.*

Proposition 7. *A predicate p is satisfiable w.r.t. a fKB $\langle \Sigma, P \rangle$ iff p is satisfiable w.r.t. $\Phi(\Sigma) \cup P$.*

Proposition 7 also holds for satisfiability checking of concept expressions C: introduce a rule $p(X) \leftarrow C(X)$ in P and check satisfiability of p.

Using the translation from fKBs to FoLPs in Proposition 7 and the polynomiality of this translation (Proposition 6), together with the complexity of the algorithm for satisfiability checking w.r.t. FoLPs, we have the following result:

Proposition 8. *Satisfiability checking w.r.t. fKBs is in* 2-NEXPTIME.

As satisfiability checking of $\mathcal{ALC}$ concepts w.r.t. an $\mathcal{ALC}$ TBox (note that $\mathcal{ALC}$ is a fragment of $\mathcal{SHOQ}$) is EXPTIME-complete ([1, Chapter 3]), we have that satisfiability checking w.r.t. fKBs is EXPTIME-hard.

Proposition 9. *Satisfiability checking w.r.t. fKBs is* EXPTIME*-hard.*

6 Discussion and Outlook

We compare fKBs to the r-hybrid KBs from [21], which extends $\mathcal{DL}$+*log* from [20] with inequalities and negated DL atoms.

In [21], a r-hybrid KB consists of a DL knowledge base and a disjunctive Datalog program where each rule is *weakly DL-safe*: (a) every variable in the rule appears in a positive atom in the body of the rule (*Datalog safeness*), and (b) every variable either occurs in a positive non-DL atom in the body of the rule, or it only occurs in positive DL atoms in the body of the rule.

The semantics of r-hybrid and fKBs correspond to a large extent. A main difference is that fKBs do not make the *standard names assumption*, in which basically the domain of every interpretation is the same infinitely countable set of constants. Some more key differences to note:

- We do not require Datalog safeness neither do we require weakly DL-safeness. Indeed, fKBs may have a rule component that is not weakly DL-safe. Take the fKB $\langle \Sigma, P \rangle$ from Example 2 with P. The atom $\mathit{Father}(X)$ is a DL-atom s. t. the rule is neither Datalog safe nor weakly DL-safe. Modifying the program to $\mathit{unhappy}(X) \leftarrow \mathit{Human}(X), \mathit{not}\ \mathit{Father}(X)$ leads to a Datalog safe program, however, it is still not weakly DL-safe as X is not appearing only in positive DL-atoms. On the other hand, both the above rules are FoLPs and thus constitute a valid component of an fKB.
- For r-hybrid KBs, due to the safeness conditions, it suffices for satisfiability checking to ground the rule component with the constants appearing explicitly in the KB.[4] One does not have such a property for fKBs. Consider the fKB $\langle \Sigma, P \rangle$ with $\Sigma = \emptyset$ and the program P with rules $a(X) \leftarrow \mathit{not}\ b(X)$ and $b(0) \leftarrow$. This program is a FoLP, but it is not Datalog safe nor is it weakly DL-safe. Grounding only with the constants in the program yields that a is not satisfiable; however, grounding with, e.g., $\{0, x\}$, makes a satisfiable.

[4] [21, 20] considers checking satisfiability of KBs rather than satisfiability of predicates. However, the former can easily be reduced to the latter.

– Decidability for satisfiability checking of r-hybrid KBs is guaranteed if decidability of the conjunctive query containment problem is guaranteed for the DL at hand. In contrast, we relied on a translation of DLs to FoLPs for establishing decidability, and not all DLs can be translated this way; we illustrated the translation for $\mathcal{SHOQ}$.

Hybrid MKNF KBs [17, 16] consist of a DL component and a component of so-called *MKNF* rules. Such MKNF rules allow for modal operators **K** and **not** in front of atoms; for a detailed definition, we refer the reader to [17]. Hybrid MKNF KBs generalize approaches to integrating ontologies and rules such as CARIN [15], $\mathcal{A}$-log [2], DL-safe rules [18], and the Semantic Web Rule Language (SWRL) [13], as well as r-hybrid KBs [21]. In particular, one can write the latter as equisatisfiable hybrid MKNF KBs [17, Theorem 4.8].

In contrast with fKBs, hybrid MKNF KBs, like r-hybrid KBs, make the standard names assumption. Every variable in a rule has to appear in a non-DL atom of the body of the rule. As with r-hybrid KBs, our fKBs do not have such a restriction of the interaction between the structural DL component and the rule component, but rely instead on the existence of an integrating framework (FoLPs under an open answer set semantics) that has reasoning support; reasoning support that we provided in this paper.

A formalism related to FoLPs is $\mathbb{FDNC}$ [22]. $\mathbb{FDNC}$ is an extension of ASP with function symbols where rules are syntactically restricted in order to maintain decidability. While the syntactical restriction is similar to the one imposed on FoLP rules, predicates having arity maximum two, and the terms in a binary literal can be seen as arcs in a forest (imposing the Forest Model Property), the direction of deduction is different: while for FoLPs, all binary literals in a rule body have an identical first term which is also the term which appears in the head, for $\mathbb{FDNC}$ (with the exception of one rule type) the second term is the one which also appears in the head. $\mathbb{FDNC}$ rules are required to be safe unlike FoLP ones. The complexity for standard reasoning tasks for $\mathbb{FDNC}$ is EXPTIME-complete and worst-case optimal algorithms are provided.

For the future, we intend to look into an extension of f-hybrid knowledge bases and its reasoning algorithm, from $\mathcal{SHOQ}$ towards $\mathcal{SROIQ}(\mathbf{D})$ the DL underlying OWL-DL in OWL 2[5]. A prototype implementation of the algorithm is planned, and will feed the need for optimization strategies.

References

1. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The DL Handbook: Theory, Implementation, and Applications*, 2003.
2. F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *J. of Intelligent and Cooperative Information Systems*, 10:227–252, 1998.

[5] http://www. w3.org/2007/OWL

3. T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, 172(12-13):1495–1539, 2008.
4. C. Feier and S. Heymans. Hybrid reasoning with forest logic programs. Technical Report INFSYS RESEARCH REPORT 184-08-14, KBS Group, Vienna University of Technology, Austria, December 2008. www.kr.tuwien.ac.at/staff/heymans/priv/projects/fwf-doasp/eswc2009-tr.pdf.
5. C. Feier and S. Heymans. A sound and complete algorithm for simple conceptual logic programs. In *Proc. of ALPSWS 2008*, 2008.
6. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of ICLP'88*, pages 1070–1080, 1988.
7. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *Proc. of the World Wide Web Conference (WWW)*, pages 48–57, 2003.
8. S. Heymans, J. de Bruijn, L. Predoiu, C. Feier, and D. Van Nieuwenborgh. Guarded hybrid knowledge bases. *TPLP*, 8(3):411–429, 2008.
9. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Conceptual logic programs. *Annals of Mathematics and Artificial Intelligence (Special Issue on Answer Set Programming)*, 47(1–2):103–137, 2006.
10. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Open answer set programming for the semantic web. *J. of Applied Logic*, 5(1):144–169, 2007.
11. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Open answer set programming for the semantic web. *J. of Applied Logic*, 5(1):144–169, 2007.
12. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Open answer set programming with guarded programs. *ACM Trans. on Comp. Logic*, 9(4), October 2008.
13. I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL rules language. In *Proc. of the World Wide Web Conference (WWW)*, pages 723–731. ACM, 2004.
14. M. Krötzsch, S. Rudolph, and P. Hitzler. Description logic rules. In *Proc. 18th European Conf. on Artificial Intelligence(ECAI-08)*, pages 80–84, 2008.
15. A. Y. Levy and M. Rousset. CARIN: A Representation Language Combining Horn Rules and Description Logics. In *Proc. of ECAI'96*, pages 323–327, 1996.
16. B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and logic programming live together happily ever after? In *Proc. of ISWC*, pages 501–514, 2006.
17. B. Motik and R. Rosati. Closing Semantic Web Ontologies. Technical report, 2006.
18. B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.
19. R. Rosati. On the decidability and complexity of integrating ontologies and rules. *Web Semantics*, 3(1):41–60, 2005.
20. R. Rosati. DL+log: Tight integration of description logics and disjunctive datalog. In *Proc. of the Int. Conf. on Principles of KR and Reas. (KR)*, pages 68–78, 2006.
21. R. Rosati. On combining description logic ontologies and nonrecursive datalog rules. In *Proc. of the 2nd Int. Conf. on Web Reasoning and Rule Systems*, 2008.
22. M. Simkus and T. Eiter. Fdnc: Decidable non-monotonic disjunctive logic programs with function symbols. In *Proc. 14th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2007)*, 2007.
23. M. Smith, C. Welty, and D. McGuinness. OWL Web Ontology Language Guide. `http://www.w3.org/TR/owl-guide/`, 2004.
24. S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH-Aachen, 2001.
25. M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th Int. Colloquium on Automata, Languages and Programming*, pages 628–641, 1998.