Integrating Ontology Languages and Answer Set Programming

S. Heymans D. Vermeir

Dept. of Computer Science
Vrije Universiteit Brussel, VUB
{*sheymans,dvermeir*}*@vub.ac.be*

Abstract

We integrate ontology languages and logic programming (LP) by extending disjunctive logic programs (DLPs) and their semantics in order to support inverses and an infinite universe, without introducing function symbols. We show that this extension is still decidable, and can be used to simulate, on the one hand, answer set programming with a finite universe, and on the other hand, several expressive description logics (DLs), which can be seen as ontology languages. The integration leads to a "best of both worlds": from the LP side it inherits a flexible and intuitive representation of knowledge, whereas the DLs side provides the possibility to represent infinite knowledge.

1. Introduction

Ontologies and the languages used to represent them are becoming increasingly more important in knowledge representation areas such as the "Semantic Web" [4] where they provide an agreed and shared understanding [15] of certain domains. The formal semantics of those languages can be given by DLs [2], which can also provide the procedures necessary to reason with the ontological knowledge. Another knowledge representation formalism that has been around for a while are the several logic programming paradigms, of which we consider answer set programming [7, 13].

In this paper we attempt to integrate both representation formalisms, by allowing inverses and infinite domains in the answer set programming semantics, without introducing function symbols. However, just extending the semantics, and allowing for arbitrary infinity, leads to undecidable reasoning procedures. We therefore restrict DLPs to free tree DLPs, which basically are DLPs where the rules have a tree structure. These free tree DLPs prove to be highly flexible, and can express diverse knowledge in a natural way. Assume for example that students that get high scores are either smart or work hard and have a smart friend that wants to help. The corresponding program would consequently consist of the two rules:$highScore(X) \leftarrow worksHrd(X), friend(X,Y), wantsHelp(X,Y), smart(Y)$ and $highScore(X) \leftarrow smart(X)$. Compare this with a possible translation to a DL axiom: $highScore = smart \sqcup (worksHrd \sqcap \exists(friend \sqcap wantsHelp).smart)$. Not only is this less appealing and less intuitive to most software engineers than the free tree DLP but it also assumes the existence of role intersection, a feature absent in most DLs.

We show that certain DLs can be simulated with free tree DLPs. The attempt to simulate DLs in a logic programming formalism is not new. In [1] the DL $\mathcal{ALCQI}$ is successfully translated into a DLP. However, to take into account infinite interpretations [1] presumes, for technical reasons, the existence of function symbols. Our approach does not make such assumptions; the ability for reasoning with infinite knowledge is built into the semantics of free tree DLPs. In fact our approach is more general and actually subsumes the one in [1]. Whereas we extend answer set programming to take into account infinity, and thus are able to simulate very expressive DLs, in [8] DLs are restricted to make a translation possible into (and from) definite equality-free Datalog logic programs (*def-LPs*). For example, disjunction can only appear on the left-hand side of DLs axioms, so as to make the translation into def-LP possible. Whereas this approach has the advantage that reasoning in DLs can be achieved through simple Datalog, it has the disadvantage that one loses the expressive possibilities of DLs.

In [6] the main focus is not on the simulation of DLs with LP paradigms, but on a coexistence of DLs with, mostly, Datalog. The idea behind these *hybrid systems* is to allow for the two different knowledge representation formalisms to retain their own strong points, while taking profit of the benefits of the other one. We pursue a total and "real" integration (not just a "living apart together" integration), by defining an extension of answer set programming, in which particular DLs can be simulated.

The remainder of this paper is organized as follows: Section 2 extends the answer set programming semantics to

take into account infinite domains. Section 3 restricts the programs to DLPs with a tree structure in order to enforce decidability. A simulation of a particular DL and a discussion of the expressiveness of free tree DLPs can be found in Sect. 4. Finally, Sect. 5 contains conclusions and directions for further research. All proofs can be found in [9].

2. Answer Set Programming with Infinity

We give some basic definitions about disjunctive logic programs (DLPs) and answer sets [7, 13] , and extend them to take into account infinite domains and inverses. Both extensions are inspired by DLs, and, more generally, the need to represent and reason with possibly infinite knowledge.

We call individual names *constants* and write them as lowercase letters, *variables* will be denoted with uppercase letters. Variables and constants are *terms*. *Atoms* are defined as being of the form $p_1(t_1)$, $p_2(t_1, t_2)$, $p_2{}^-(t_1, t_2)$, with p_1 a unary predicate, and p_2 a binary predicate, t_1 and t_2 are terms. Indeed, we restrict to unary and binary predicates; inverting atoms does not seem to make sense for predicates of greater arity.

A *literal* is an atom or an atom preceded by $\neg$, i.e. l is a literal if $l = a$ or $l = \neg a$ for an atom a. An *extended literal* is a literal l or something of the form $not(l)$, with l a literal. Literals, or atoms, not containing variables are *ground*. For a set X of literals, $\neg X = \{\neg l \mid l \in X\}$, where we define $\neg\neg a$ as a. A set of ground literals X is *consistent* if $X \cap \neg X = \emptyset$. For a set X of extended literals, we define $X^- = \{l \mid not(l) \in X\}$, i.e. the set of underlying literals. Furthermore, we assume the existence of a binary predicate $\neq$, with the usual interpretation.

A *disjunctive logic program* (DLP) is a finite set of rules $\alpha \leftarrow \beta$ where α and β are finite sets of extended literals. We call programs where for each rule $\beta^- \cup \alpha^- = \emptyset$, *programs without negation as failure (naf)*. Programs without naf such that for all rules β contains at most one element, i.e. no disjunction in the head, are called *simple programs*.

Programs that do not contain variables are ground. For a program P and a (possibly infinite) non-empty set of constants $\mathcal{H}$, such that every constant appearing in P is in $\mathcal{H}$, we call $P_\mathcal{H}$ the *grounded program* obtained from P by substituting every variable in P by every possible constant in $\mathcal{H}$. Note that $P_\mathcal{H}$ may contain an infinite number of rules (if $\mathcal{H}$ is infinite). An infinite DLP must be a grounded version of a finite one.

The *universe* of a grounded program $P_\mathcal{H}$ is the (possibly infinite) non-empty set of constants $\mathcal{H}_{P_\mathcal{H}}$ appearing in $P_\mathcal{H}$. The *base* of a grounded program $P_\mathcal{H}$ is the (possibly infinite) set $\mathcal{B}_{P_\mathcal{H}}$ of ground atoms that can be constructed using the predicates in $P_\mathcal{H}$ and their inverses, with the constants in $\mathcal{H}$. An *interpretation* I of a grounded DLP P is any consistent set of literals that is a subset of $\mathcal{B}_P \cup \neg\mathcal{B}_P$. An interpretation I of a grounded DLP P without naf *satisfies* a rule $\alpha \leftarrow \beta$ if $\alpha \cap I \neq \emptyset$ whenever $\beta \subseteq I$. Or, intuitively, if the conjunction of literals in the body of a rule is true, the disjunction of the literals in the head must be true. An interpretation I is a *model* of a grounded DLP P without naf if it satisfies every rule in P and $p(t_1, t_2) \in I \iff p^-(t_2, t_1) \in I$ for all literals $p(t_1, t_2)$ in $\mathcal{B}_P \cup \neg\mathcal{B}_P$. Furthermore, it is a *minimal model* if there is no model $J \subset I$ of P.

For a grounded DLP P and an interpretation I, the Gelfond-Lifschitz transformation [13], is the program P^I, obtained by deleting in P each rule that has $not(l)$ in its body with $l \in I$, each rule that has $not(l)$ in its head with $l \notin I$, and all $not(l)$ in the bodies and heads of the remaining rules. An *interpretation* of a DLP P (not grounded) is a tuple $(I, \mathcal{H}_I)$, such that I is an interpretation of the grounded $P_{\mathcal{H}_I}$. An interpretation $(I, \mathcal{H}_I)$ of a DLP P is an *answer set* of P if I is a minimal model of $P^I_{\mathcal{H}_I}$.

A DLP P is *consistent* if P has an answer set. For a unary p (p possibly negated), appearing in P, we say that p is *satisfiable* w.r.t. P if there exists an answer set $(I, \mathcal{H}_I)$ of P such that $p(a) \in I$ for an $a \in \mathcal{H}_I$; if $\mathcal{H}_I$ is finite we call p *finitely satisfiable*. Checking this satisfiability for a (possibly negated) unary predicate is called *satisfiability checking*.

Take for example the program $youngBricoleur(X) \leftarrow child(X), make(X,T), toy(T), play(X,T)$ which expresses that young bricoleurs are children that make and play with their own toys. Note that answer sets containing the knowledge *young_bricoleur(albert)* about a particular individual, should also contain the knowledge *child(albert)*,*toy(some_toy)*,*make(albert,some_toy)*, and *play(albert,some_toy)*, by the minimality of answer sets. However the rule itself does not contain information about individuals, or does not presuppose a finite domain; there may be infinitely many bricoleurs, and for all of them the right conclusions will be derived.

3. Free Tree DLPs

Answer set programming with infinity is powerful, in fact it is too powerful. One can show that satisfiability checking is undecidable in such a general context. Indeed, we would be able to simulate an extension of the DL $\mathcal{SHIQ}$ [12], where the roles in number restrictions are not simple (i.e. roles may be transitive or have transitive subroles). In [12] it was shown that in such a DL satisfiability checking is undecidable.

We propose *free tree DLPs* as a trade-off between expressiveness and decidability of satisfiability checking. To clarify our choice, consider a scenario where we want to ask a monitoring system if there were discovered new unknown errors. Such errors are discovered if previously there

were no errors detected but now the monitored system is not running anymore, which means that a previously unidentified error has occurred and crashed the system (we assume that known errors are being dealt with in a clean manner). The monitoring system discovers no new problems, if previously everything was going fine and at present the system is still up and running. The corresponding DLP could be something like the following:

$$newErr(X) \leftarrow \neg run(X), yest(X,Y), \neg Prob(Y) \quad (1)$$
$$\neg Prob(X) \leftarrow run(X), yest(X,Y), not(Prob(Y)) \quad (2)$$
$$Prob(X) \leftarrow not(\neg Prob(X)) \quad (3)$$
$$\leftarrow yest^{-}(X,Y_1), yest^{-}(X,Y_2), Y_1 \neq Y_2 \quad (4)$$
$$yest(X,Y) \vee not(yest(X,Y)) \leftarrow \quad (5)$$
$$run(X) \vee not(run(X)) \leftarrow \quad (6)$$
$$\neg run(X) \vee not(\neg run(X)) \leftarrow \quad (7)$$

with the literal $yest(X,Y)$ expressing that if X is today, then Y is yesterday; rule (4) assures that there are no two different yesterdays for one today. We call rules of this type *functional rules*. Rules (5), (6), and (7) imply that answer sets are free to contain the corresponding predicates or not, thus we call them *free rules*. Rules like (1), (2), and (3) are *tree rules*, i.e. they can be syntactically viewed as trees. Before defining free tree DLPs we define (finite) trees.

A *(finite) tree* T is a (finite) subset of $\mathbb{N}_0^*$ ($\mathbb{N}_0 = \mathbb{N} \setminus \{0\}$) such that if $x \cdot c \in T$ for $x \in \mathbb{N}_0^*$ and $c \in \mathbb{N}_0$, we have that $x \in T$. More formally we define free tree DLPs as follows.

Definition 1 *A free tree DLP is a DLP that does not contain constants and such that every rule is of one of the following types:*

- *free rule* $a \vee not(a) \leftarrow$ *with* a *a literal.*
- *constraint* $\leftarrow f(X,Y), b(Y), not(g(X,Y)), not(d(Y))$ *with literals* $f(X,Y), b(Y), g(X,Y)$ *and* $d(Y)$*. Not all literals have to appear in the body, but if* $not(g(X,Y))$ *or* $not(d(Y))$ *is present,* $f(X,Y)$ *has to be too.*
- *constraints* $a(X) \leftarrow f(X,Y_1), f(X,Y_2), Y_1 \neq Y_2$ *with* $f(X,Y_1)$ *and* $f(X,Y_2)$ *literals, and the head possibly empty. If the head is empty we call* f *functional.*
- *binary rules* $f(X,Y) \leftarrow a_1(X), \ldots, a_m(X), not(b_1(X)), \ldots, not(b_n(X)), f_1(X,Y), \ldots, f_o(X,Y), c_1(Y), \ldots, c_p(Y), not(d_1(Y)), \ldots, not(d_q(Y))$ *with at least one* $f_i(X,Y)$ *in the body.*
- *tree rules* $a(X) \leftarrow \beta$ *with* $a(X)$ *a literal and* β *a finite set of extended literals with the following restrictions:*
  - *there exists a finite tree* T *such that there is a bijection* $\phi : T \rightarrow Vars$*, with* $Vars$ *the variables in* $a(X) \leftarrow \beta$*, such that* y *is a successor of* x *in* T *iff there exists a literal* $f(\phi(x), \phi(y))$ *in* β*,*
  - *"not" appears only in front of unary literals, i.e. all literals in* β^- *are unary.*

The monitoring system example is a free tree DLP, and, furthermore, $newErr$ is satisfiable, but not finitely satisfiable. Indeed $(M, \{a_0, a_1, \ldots\})$ with $M = \{newErr(a_0), yest(a_0,a_1), \neg run(a_0), yest^{-}(a_1,a_0), Prob(a_0), \neg Prob(a_1), yest(a_1,a_2), run(a_1), yest^{-}(a_2,a_1), \neg Prob(a_2), yest(a_2,a_3), run(a_2), yest^{-}(a_3,a_2), \ldots\}$ is an infinite answer set of the program and there is no answer set that is finite and satisfies $newErr$. Intuitively, this answer set says that at time a_0 there has occurred a totally new error, since at all earlier times no problems were detected. This result could not have been attained with traditional finite answer set programming.

To show that satisfiability checking in free tree DLPs is decidable, (this is why we restricted the DLPs in the first place) we use techniques similar to the ones used for μ-calculus with backward modalities [18], and already successfully applied to several DLs [5]. Those techniques involve the reduction of satisfiability checking to checking non-emptiness of a two-way alternating automaton (2ATA) [18], which is decidable. 2ATA are automata on infinite trees, such that it comes as no surprise that the decidability of free tree DLPs, as is the case for the μ-calculus and related modal logics, comes down to the tree-model property [17]. This property claims that if a predicate is satisfiable it is satisfiable by a model that has a tree structure. Free tree DLPs have the tree-model property and mainly owe this to the syntactic tree-structure of tree rules.

Theorem 1 *Satisfiability checking w.r.t. a free tree DLP is decidable.*

Free tree DLPs are still general enough to simulate several DLs, as we will show in the next section, but can also simulate finite answer set programming for programs without disjunction in the head (we omit the "disjunctive" qualifier and call them *logic programs*), and possibly predicates of greater arity.

Theorem 2 M *is an answer set of a logic program* P *iff* $(M', \{a\})$ *is an answer set of the free tree DLP* P'*, with some constant* a *,* $M' = \{l(a) | l \in M\}$ *and* $P' = \{r(X) | r \in P^M_{\mathcal{H}_M}\}$*, with* $r(X)$ *defined such that every literal* l *in* r *is replaced by* $l(X)$*.*

This theorem implies that whatever DL is simulated by means of finite answer set programming, it can be simulated by a free tree DLP.

4. Simulating Description Logics

Description logics play an important role in the evolution of ontology languages such as DAML+OIL [3] and more recently OWL [16], as they can be used to express the formal semantics of those languages. The DL $\mathcal{SHIQ}$ corresponds for example to the ontology language OIL [10].

In this section we consider the slightly less expressive DL $\mathcal{SHIF}$ [11], and instead of transitive roles we allow for transitive closure of roles. We denote this particular DL as $\mathcal{SHIF}^*$. $\mathcal{SHIF}^*$ can be easily simulated by an intuitive and elegant free tree DLP translation.

We define the syntax of $\mathcal{SHIF}^*$ concept expressions as follows.

$$D_1, D_2 \rightarrow A|\neg D_1|D_1 \sqcap D_2|D_1 \sqcup D_2|\exists R.D_1|\forall R.D_1|(\leq 1Q)$$

and Q is a P or a P^-, R is Q or Q^* with A a concept name and P a role name. We take $(\geq 2Q)$ to be a shorthand for $\neg(\leq 1Q)$. The semantics of a $\mathcal{SHIF}^*$ concept expression is given by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ which consists of a non-empty (possibly infinite) domain $\Delta^{\mathcal{I}}$, and an interpretation function $\cdot^{\mathcal{I}}$ defined as follows.

$$\begin{aligned}
A^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \text{ for concept names } A \\
P^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \text{ for role names } P \\
P^{-\mathcal{I}} &= \{(y,x)|(x,y) \in P^{\mathcal{I}}\} \text{ for role names } P \\
(\neg D_1)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus D_1^{\mathcal{I}} \\
(D_1 \sqcap D_2)^{\mathcal{I}} &= D_1^{\mathcal{I}} \cap D_2^{\mathcal{I}} \\
(D_1 \sqcup D_2)^{\mathcal{I}} &= D_1^{\mathcal{I}} \cup D_2^{\mathcal{I}} \\
(\exists R.D_1)^{\mathcal{I}} &= \{x|\exists y : (x,y) \in R^{\mathcal{I}} \wedge y \in D_1^{\mathcal{I}}\} \\
(\forall R.D_1)^{\mathcal{I}} &= \{x|\forall y : (x,y) \in R^{\mathcal{I}} \Rightarrow y \in D_1^{\mathcal{I}}\} \\
(\leq 1Q)^{\mathcal{I}} &= \{x|\#\{y|(x,y) \in Q^{\mathcal{I}}\} \leq 1\} \\
(R^*)^{\mathcal{I}} &= R^{\mathcal{I}^*} \text{ i.e. the reflexive transitive closure of } R^{\mathcal{I}}
\end{aligned}$$

A *terminological axiom* is of the form $C_1 \sqsubseteq C_2$, with C_1 and C_2 arbitrary concept expressions. An interpretation $\mathcal{I}$ satisfies a terminological axiom $C_1 \sqsubseteq C_2$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$. A *role axiom* is of the form $R_1 \sqsubseteq R_2$, with R_1 and R_2 roles (possibly inverted or transitively closed). An interpretation $\mathcal{I}$ satisfies a role axiom $R_1 \sqsubseteq R_2$ if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$. A knowledge base Σ is a set of terminological and role axioms. An interpretation $\mathcal{I}$ is a *model* of Σ if $\mathcal{I}$ satisfies every axiom in Σ. A $\mathcal{SHIF}^*$ concept expression C is *satisfiable* w.r.t. Σ if there exists a model $\mathcal{I}$ of Σ such that C has a non-empty interpretation, i.e. $C^{\mathcal{I}} \neq \emptyset$.

We define the *closure* $clos(C, \Sigma)$ of a concept expression C and the $\mathcal{SHIF}^*$ knowledge base Σ, as follows.

- for every concept expression D in $\{C\} \cup \Sigma$ we have $D \in clos(C, \Sigma)$,
- for every D in $clos(C, \Sigma)$, we have
  1. if $D = \neg D_1$ then $D_1 \in clos(C, \Sigma)$,
  2. if $D = D_1 \sqcup D_2$ then $\{D_1, D_2\} \subseteq clos(C, \Sigma)$,
  3. if $D = D_1 \sqcap D_2$ then $\{D_1, D_2\} \subseteq clos(C, \Sigma)$,
  4. if $D = \exists R.D_1$ then $\{R, D_1\} \subseteq clos(C, \Sigma)$,
  5. if $D = \forall R.D_1$, then $\{D_1, \exists R.\neg D_1\} \subseteq clos(C, \Sigma)$,
  6. if $D = \leq 1Q$, then $\{Q, (\geq 2Q)\} \subseteq clos(C, \Sigma)$,
  7. for all $R^* \in clos(C, \Sigma)$, then $R \in clos(C, \Sigma)$,
  8. for all $D \in clos(C, \Sigma)$, $\neg D \in clos(C, \Sigma)$.

It is straightforward to simulate satisfiability checking in $\mathcal{SHIF}^*$ with free tree DLPs. We define $\Phi(C, \Sigma)$ to be the free tree DLP, obtained from Σ and C as in Table 1.

Table 1. $\mathcal{SHIF}^*$ simulation

$clos(C, \Sigma)$	$\Phi(C, \Sigma)$
concepts A	$P_A(X) \vee not(P_A(X)) \leftarrow$
role names P	$P_P(X,Y) \vee not(P_P(X,Y)) \leftarrow$
	$\neg P_P(X,Y) \vee not(\neg P_P(X,Y)) \leftarrow$
roles P^-	$P_{P^-}(a,b) \leftarrow P_P{}^-(a,b)$
	$\neg P_{P^-}(a,b) \leftarrow \neg P_P{}^-(a,b)$
expressions D	$\neg P_D(X) \leftarrow not(P_D(X))$
$D = \neg E$	$P_{\neg E}(X) \leftarrow \neg P_E(X)$
$D = E \sqcap F$	$P_{E \sqcap F}(X) \leftarrow P_E(X), P_F(X)$
$D = E \sqcup F$	$P_{E \sqcup F}(X) \leftarrow P_E(X)$
	$P_{E \sqcup F}(X) \leftarrow P_F(X)$
$D = \exists Q.E$	$P_{\exists Q.E}(X) \leftarrow P_Q(X,Y), P_E(Y)$
$D = \exists Q^*.E$	$P_{\exists Q^*.E}(X) \leftarrow P_E(X)$
	$P_{\exists Q^*.E}(X) \leftarrow P_Q(X,Y), P_D(Y)$
$D = \forall R.E$	$P_{\forall R.E}(X) \leftarrow \neg P_{\exists R.\neg E}(X)$
$D = \leq 1Q$	$P_{\leq 1Q}(X) \leftarrow not(P_{\geq 2Q}(X))$
	$P_{\geq 2Q}(X) \leftarrow P_Q(X,Y_1), P_Q(X,Y_2), Y_1 \neq Y_2$
$C_1 \sqsubseteq C_2 \in \Sigma$	$\leftarrow P_{C_1}(X), not(P_{C_2}(X))$
$R_1 \sqsubseteq R_2 \in \Sigma$	$\leftarrow P_{R_1}(X,Y), not(P_{R_2}(X,Y))$

Take for example a $\mathcal{SHIF}^*$ knowledge base consisting of the single axiom $ProbChild \sqsubseteq \exists parent.ProbChild.$ expressing that problem children tend to have parents that were themselves problem children. Translating this to free tree DLP gives us the rules that simulate explicit DLs behavior:

$$\begin{aligned}
P_{ProbChild(X)} \vee not(P_{ProbChild(X)}) &\leftarrow \\
\neg P_{ProbChild(X)} &\leftarrow not(P_{ProbChild(X)}) \\
P_{parent(X,Y)} \vee not(P_{parent(X,Y)}) &\leftarrow \\
\neg P_{parent(X,Y)} \vee not(\neg P_{parent(X,Y)}) &\leftarrow \\
P_{\exists parent.ProbChild}(X) &\leftarrow P_{parent}(X,Y), P_{ProbChild}(Y)
\end{aligned}$$

and of course the translation of the axiom itself

$$\leftarrow \quad P_{ProbChild}(X), not(P_{\exists parent.ProbChild}(X))$$

The obtained $\Phi(C, \Sigma)$ is indeed a free tree DLP and furthermore we can check that satisfiability checking in $\mathcal{SHIF}^*$ is equivalent to satisfiability checking in this free tree DLP.

Theorem 3 *A $\mathcal{SHIF}^*$ concept expression C is satisfiable w.r.t. a $\mathcal{SHIF}^*$ knowledge base Σ iff $P_C(X)$ is satisfiable w.r.t. $\Phi(C, \Sigma)$.*

A more interesting example is, is for example to check the satisfiability of $\neg C \sqcap \exists F^-.(C \sqcap (\leq 1\ F\ )) \sqcap \forall F^{-*}.(\exists F^-.(C \sqcap (\leq 1\ F\ )))$. As noted in [12], each of the models that make this concept expression satisfiable have

an infinite domain (in [12], transitivity is used instead of transitive closure). By Theorem 3 the corresponding predicate is satisfiable. In correspondence with the DLs case it can be checked that all answer sets that make the predicate satisfiable are infinite.

Another example expresses a specification of a "leads" predicate, i.e. a person "rules" something if he is a president and he leads a country: $rules(X,Y) \leftarrow president(X), leads(X,Y), country(Y)$.

It is possible to express in $\mathcal{SHIF}^*$ that, if x rules y then x leads y, and x must be a president, y a country. However, the other way around, "if x leads y and y is a country, x is a president then x rules y" is inexpressible in $\mathcal{SHIF}^*$.

While the argument that free tree DLPs are more intuitive than DLs is a rather subjective one, this simple example shows that free tree DLPs are more expressive than $\mathcal{SHIF}^*$, as well as some aspects of the more expressive DL $\mathcal{SHOIQ}(\mathbf{D})$, which is the DL corresponding to the ontology language DAML+OIL [8].

5. Conclusions and Directions for Further Research

We have extended answer set programming with infinity, and provided a decidable restriction of disjunctive logic programs, by enforcing a tree structure on the rules. Not only is this extension more intuitive than DLs, it can also simulate a large class of DLs, and is more expressive than most of them. The extension effectively integrates ontology languages with answer set programming, and allows for representing and reasoning with knowledge in an intuitive, expressive and unified way.

Interesting directions for further research are to extend free tree DLP such that it allows the simulation of more expressive DLs. Similar to [14] we could extend answer set programming with constraint rules that enforce a certain cardinality, this would allow to simulate DLs with expressive number restrictions (not just functional ones). Allowing constants to appear in the free tree DLPs would make the simulation possible of DLs with individuals as $\mathcal{SHOIQ}(\mathbf{D})$, which is the description logic corresponding to DAML+OIL. Since we are able to express infinite knowledge, it would be interesting to see how well free tree DLPs scale for temporal reasoning or other reasoning paradigms explicitly depending on a notion of infinity.

References

[1] G. Alsaç and C. Baral. Reasoning in description logics using declarative logic proramming. `http://www.public.asu.edu/~guray/dlreasoning.pdf`, 2002.

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

[3] S. Bechhofer, C. Goble, and I. Horrocks. DAML+OIL is not enough. In *Proceedings of the First Semantic Web Working Symposium (SWWS'01)*, pages 151–159. CEUR, 2001.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, pages 34–43, May 2001.

[5] D. Calvanese, G. D. Giacomo, and M. Lenzerini. 2ATAs make DLs easy. In *Proc. of the 2002 Description Logic Workshop (DL'02)*, 2002.

[6] F. M. 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.

[7] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[8] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of Twelfth International World Wide Web Conference (WWW 2003)*, 2003. To appear.

[9] S. Heymans and D. Vermeir. Integrating ontology languages and answer set programming. Technical report, Vrije Universiteit Brussel, Dept. of Computer Science, 2003.

[10] I. Horrocks. A denotational semantics for Standard OIL and Instance OIL. `http://www.ontoknowledge.org/oil/downl/semantics.pdf`, 2000.

[11] I. Horrocks and U. Sattler. A description logic with transitive and converse roles and role hierarchies. LTCS-Report 98-05, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1998.

[12] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705, pages 161–180. Springer-Verlag, 1999.

[13] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[14] P. Simons. Extending the stable model semantics with more expressive rules. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 305–316. Springer-Verlag, 1999.

[15] M. Uschold and M. Grüninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[16] F. van Harmelen, J. Hendler, I. Horrocks, and L. A. S. D. L. McGuinness, P. F. Patel-Schneider. Web Ontology Language (OWL) Reference Version 1.0. W3C Working Draft - `http://www.w3.org/TR/owl-ref/`, February 2003.

[17] M. Y. Vardi. Why is modal logic so robustly decidable? Technical Report TR97-274, Rice University, Apr. 12, 1997.

[18] M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of the 25th Int. Coll. on Automata, Languages and Programming*, pages 628–641. Springer, 1998.