Integrating Description Logics and Answer Set Programming

S. Heymans and D. Vermeir*

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

Abstract. We integrate an expressive class of description logics (DLs) and answer set programming by extending the latter to support inverted predicates and infinite domains, features that are present in most DLs. The extended language, conceptual logic programming (CLP) proves to be a viable alternative for intuitively representing and reasoning nonmonotonically, in a decidable way, with possibly infinite knowledge. Not only can conceptual logic programs (CLPs) simulate finite answer set programming, they are also flexible enough to simulate reasoning in an expressive class of description logics, thus being able to play the role of ontology language, as well as rule language, on the Semantic Web.

1 Introduction

Description logics (DLs) [2] and answer set programming [10, 19] are well-established knowledge representation mechanisms. We integrate them by adding predicate inverses to disjunctive logic programs (DLPs) and extending the answer set semantics by allowing for an infinite domain, without introducing function symbols. Both extensions to answer set programming are inspired by their presence in most DLs, effectively integrating the flexible and intuitive way of representing knowledge in logic programming with DLs features, making elegant reasoning with infinite knowledge possible.

However, simply extending answer set programming leads to undecidability, notably of satisfiability checking. We therefore restrict the syntactic structure of DLPs, obtaining conceptual logic programs (CLPs). Satisfiability checking can then be decided by reducing it to checking satisfiability w.r.t. simpler DLPs with finite answer set programming techniques.

CLPs can simulate (disjunction-free) logic programs as well as expressive classes of DLs, such as $\mathcal{SHIQ}^*$. $\mathcal{SHIQ}^*$ is a slight modification of $\mathcal{SHIQ}$ [17] with transitive closure of roles instead of transitivity of roles. $\mathcal{SHIQ}$ is regarded as the formal specification of the ontology language OIL [15, 8], which can be used to express ontologies[1] on the Semantic Web [4]. Other, more expressive, ontology languages are, for

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

[1] Like DL knowledge bases or database schema's, ontologies are models of a domain, providing an *agreed* and *shared* understanding [20].

example, DAML+OIL [3] and, more recently, OWL [21] which also include support for data types and individuals.

Although, satisfiability checking w.r.t. a $\mathcal{SHIQ}^*$ knowledge base can be intuitively reduced to satisfiability checking w.r.t. a CLP the reverse is not true, i.e. there are CLP rules that cannot be translated to $\mathcal{SHIQ}^*$. Moreover, we believe that, in many cases, CLPs are more intuitive, modular and easier to use than description logics.

Consider the following example,

$$restore(X) \leftarrow crash(X), yest(X,Y), BackupSucceeded(Y) \quad (1)$$

$$BackupSucceeded(X) \leftarrow \neg crash(X), yest(X,Y), not(BackupFailed(Y)) \quad (2)$$

$$BackupFailed(X) \leftarrow not(BackupSucceeded(X)) \quad (3)$$

$$\leftarrow yest^-(X,Y_1), yest^-(X,Y_2), Y_1 \neq Y_2 \quad (4)$$

$$\leftarrow yest(X,Y_1), yest(X,Y_2), Y_1 \neq Y_2 \quad (5)$$

$$yest(X,Y) \vee not(yest(X,Y)) \leftarrow \quad (6)$$

$$crash(X) \vee not(crash(X)) \leftarrow \quad (7)$$

$$\neg crash(X) \vee not(\neg crash(X)) \leftarrow \quad (8)$$

where 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 (1) and (2) express the above knowledge, and are called tree rules, due to their tree-like structure, i.e. a tree with root X and leaf Y connected trough *yest*. Rules (4) and (5) ensure that for a particular today there can be only one yesterday and only one tomorrow, where $yest^-$ denotes the inverse of $yest$. Both rules also have a tree structure (with root X and leafs Y_1 and Y_2), and, since the conclusion part of the rule is empty, we call them tree constraints. The last three rules are so-called free rules and express that on any day a crash may or may not have occurred. In general, free rules express that certain facts may freely be added to the model, subject to restrictions implied by other rules.

The main point of attention in this example is that all answer sets, claiming a restore on a particular date, should also assure that on all previous dates the backup succeeded, explicitly demanding for an infinite domain, and an infinite domain only. Furthermore, reasoning with CLPs is clearly nonmonotonic due to the presence of *negation as failure*, i.e. the "not" in front of literals.

The attempt to integrate DLs with logic programming is not new. [1] presents, without considering decidability issues, a translation from the DL $\mathcal{ALCQI}$ to answer set programs, using, unlike in the present approach, artificial function symbols to accommodate for an infinite Herbrand universe. [11] simulates reasoning in DLs through simple datalog programs. This necessitates heavily restricting the usual DL constructors: e.g. negation or number restrictions cannot be expressed. While our approach can express those constructions and, as such, makes the possible interweaving of ontologies and rules more complete, the approach in [11] has the advantage that existing LP-reasoning engines can be used. An alternative approach is to simply add datalog-like programs to coexist with DL theories, as in [6, 7], thus exploiting the strengths of both knowledge representation mechanisms. This contrasts with our approach which aims to import the advantages of DLs into an extension of answer set programming.

Other approaches that connect rules to ontologies are, for example, [9], where a mapping of a set of descriptions in the languages RDF, RDF-S or DAML+OIL to first-order predicate calculus is specified, or [18], where DAMLJessKB enables the querying of information in DAML files, by using Jess as a forward chaining production system.

The remainder of this paper is organized as follows: Section 2 extends the answer set programming semantics to support inverses and infinite domains. Section 3 restricts the programs to DLPs with a tree structure in order to enforce the tree-model property and decidability of reasoning with CLPs. A simulation of finite answer set programming and a particular expressive DL can be found in Sect. 4. Sect. 5 contains conclusions and directions for further research.

2 Answer Set Programming with Infinity

We give some basic definitions about disjunctive logic programs (DLPs) and answer sets [10, 19], and extend them to take into account infinite domains and inverses.

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. We assume p^{--} to be defined as p for a binary predicate p, and for atoms a, we assume a^- is a with the predicate and the arguments inverted in case of binary (possibly inverted) predicates, i.e. $p(t_1, t_2)^- = p^-(t_2, t_1)$, and $p(t_1)^- = p(t_1)$ for unary predicates. Note that we restrict to unary and binary predicates; inverting atoms does not seem to make sense for predicates of greater arity.

Ground atoms are atoms without variables. 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. We define $(\neg a)^-$ as $\neg(a^-)$ for an atom a. A *ground literal* is a literal without variables. An *extended literal* is a literal l or something of the form $not(l)$, with l a literal. A *ground extended literal* is an extended literal without variables. 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.

We use Greek letters to represent sets of (unary or binary, possibly negated and/or inverted) predicates. Attaching variables then allows us to write e.g. $\alpha(X)$ for $\{a(X) | a \in \alpha\}$, $\beta(X, Y)$ for $\{b(X, Y) | b \in \beta\}$, or $not(\alpha)(X)$ for $\{not(a(X)) | a \in \alpha\}$. 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 $P_{\mathcal{H}}$. Note that $\mathcal{H}_{P_{\mathcal{H}}} = \mathcal{H}$. 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 [19], 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*.

Although we allow for infinite domains, we can motivate the presence of literals in a minimal model of a simple program in a finite way. We express the motivation of a literal more formally by means of an operator T that computes the closure of a set of literals w.r.t. a program P.

For a DLP P and an answer set $(M, \mathcal{H}_M)$ of P such that $P^M_{\mathcal{H}_M}$ is a simple program, we define the operator $T_{P^M_{\mathcal{H}_M}} : \mathcal{B}_{P^M_{\mathcal{H}_M}} \cup \neg\mathcal{B}_{P^M_{\mathcal{H}_M}} \rightarrow \mathcal{B}_{P^M_{\mathcal{H}_M}} \cup \neg\mathcal{B}_{P^M_{\mathcal{H}_M}}$ [2] as follows.

$$T_{P^M_{\mathcal{H}_M}}(B) = B \cup \{a, a^- | a \leftarrow \beta \in P^M_{\mathcal{H}_M} \wedge \beta \subseteq B\}$$

We define $T^0(B)$ as B, and $T^{n+1}(B)$ as $T^n(T(B))$. The operator gives the immediate consequences of a set B according to $P^M_{\mathcal{H}_M}$.

Theorem 1. *Let P be a DLP and $(M, \mathcal{H}_M)$ an answer set of P, with $P^M_{\mathcal{H}_M}$ a simple program. Then $\forall a \in M \cdot \exists n < \infty \cdot a \in T^n(\emptyset)$*

Proof Sketch. Assume $\exists a \in M \cdot \forall n < \infty \cdot a \notin T^n(\emptyset)$. We write down all $r : a' \leftarrow \beta \in P^M_{\mathcal{H}_M}$ with $a' = a$ or $a' = a^-$ such that $\beta \subseteq M$ and such that there exist $a_2 \in \beta$

[2] We omit the subscript if it is clear from the context.

such that $\forall n < \infty \cdot b \notin T^n(\emptyset)$. Since such r can always be chosen, we can repeat this procedure for all b ad infinitum. This way we can define a strict subset M' of M, i.e. M without a, a^- and all b with their inverses (intuitively, we throw away all the literals that are causing a to be not finitely deducible). It can be shown that M' is a model of $P^M_{\mathcal{H}_M}$. A contradiction with the minimality of M. □

The previous theorem allows to find a finite foundation for a literal in an answer set $(M, \mathcal{H}_M)$. It proves useful in the decidability proof of satisfiability checking, as well as in the DLs simulation.

3 Conceptual Logic Programs

Satisfiability checking in the above context of answer set programming with infinity is undecidable[3]. Hence we will restrict arbitrary DLPs, such that we regain the decidability of satisfiability checking while being careful so as to maintain a sufficient degree of expressiveness.

Inspired by modal logics (and DLs in particular), we restrict arbitrary DLPs to *conceptual logic programs* as to obtain DLPs such that if a unary predicate is satisfied by an answer set, it can be satisfied by an answer set with a tree structure, i.e. CLPs have the *tree-model property*. In [22] this tree-model property is held responsible for the robust decidability of modal logics. Confirming this, the tree-model property proves to be of significant importance to the decidability of satisfiability checking in CLPs.

A CLP is defined as a collection of several kinds of rules: *free rules*, i.e. rules that express that it does not matter whether a literal is in the answer set or not, provided there are no other rules prohibiting or enforcing the presence of that literal, a collection of *tree constraints*, and *tree rules*, both general rules, that are suitably restricted to ensure the tree-model property, i.e. they have a tree structure.

Formally, a *(finite) tree* T is a (finite) subset of $\mathbb{N}_0^*$[4] 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 $x \cdot c$, $c \in \mathbb{N}_0$, *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*. We call the maximum number of successors for a node in a tree, the *rank* of that tree. A *labeled tree* over an alphabet Σ is a tuple (T, V) where T is a tree and $V : T \to \Sigma$ is a function, labeling the nodes of T with elements from the alphabet. We extend the definitions of free tree DLPs from [13] by allowing for more general occurrences of inequalities, as well as the general tree structure also for constraints and rules with a binary literal in the head, instead of only for rules with a unary literal in the head.

Definition 1. *A conceptual logic program (CLP) is a set of rules that does not contain constants and such that every rule is of one of the following types:*

[3] Similar to the simulation in Section 4, it can be shown that satisfiability checking in the DL $\mathcal{SHIQ}$ [17], extended such that arbitrary roles, i.e. roles that are transitive or have transitive subroles, are allowed in number restrictions, can be reduced to checking satisfiability in an extended DLP. Since satisfiability checking for the former is undecidable [17], it remains so for the latter.

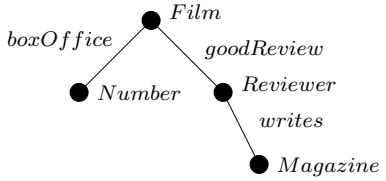
[4] $\mathbb{N}_0 = \mathbb{N} \setminus \{0\}$

- free rules $a \vee not(a) \leftarrow$ *with* a *a (binary or unary) literal. E.g. a rule such as* $p(X) \vee not(p(X)) \leftarrow$ *indicates that ground literals of the form* $p(c)$ *can be accepted (or rejected) without further motivation, subject to restrictions imposed by the other rules in the program.*
- tree rules $a(X) \leftarrow \beta$ *with* $a(X)$ *a unary 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*,*
  - *if* β *contains a literal of the form* $not(f(U, Z))$ *then* β *must also contain a positive (without "not") literal* $g(U, Z)$*,*
  - *there may be inequalities* $Y_i \neq Y_j$ *in* β *if* $\phi^{-1}(Y_1)$ *and* $\phi^{-1}(Y_2)$ *have the same predecessor in* T *(they are siblings). We call* T *the tree representation of the rule.*
- tree rules $f(X, Y) \leftarrow \beta$ *with the same tree restrictions on* β *as above, and additionally at least one positive* $g(X, Y)$ *in* β*,*
- tree constraints $\leftarrow \beta$ *with the same tree restrictions on* β *as for tree rules with a unary literal in the head.*

Consider for example the following tree rule, expressing that a top film is a film that did well at the box office and received a good review of an expert magazine.

$$topFilm(Film) \leftarrow film(Film), boxOffice(Film, Number), high(Number),$$
$$goodReview(Film, Reviewer), writes(Reviewer, Magazine), expert(Magazine)$$

Graphically, one sees that this rule has indeed a tree structure.



Note that we also allow rules of the form $a(X) \leftarrow$ in CLPs, since they can be replaced by $a(X) \vee not(a(X)) \leftarrow$ and $\leftarrow not(a(X))$. Furthermore, one does not need to have that the X in the head of a tree rule is the root of the tree representation.

In a rapidly evolving environment such as the Semantic Web, it is important to be able to revise or withdraw conclusions when additional information becomes available. Such nonmonotonicity is provided by *negation as failure*, i.e. the allowance for "not" in front of literals. Assume, for example, that we have that top films for which we cannot establish that they are released in the US are low budget films.

$$lowBudget(Film) \leftarrow topFilm(Film), not(releasedInUS(Film))$$

If x is then a top film, with nothing known about its release status, all answer sets will indicate that x is a low budget production. However, if we learn that all top films get a chance to make it also in the US, i.e. our knowledge gets enriched with

$$releasedInUS(Film) \leftarrow topFilm(Film)$$

we are no longer able to deduce that x is a low budget film.

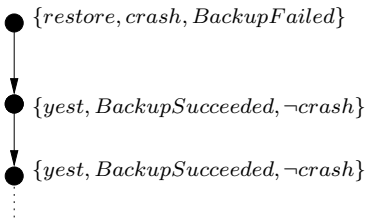
An important factor in the decidability of satisfiability checking is the assessment of the tree-model property for CLPs. We define *tree-satisfiability* as satisfiability such that the involved answer set has a tree structure. Formally, a unary predicate p (possibly negated) in a DLP P is *tree-satisfiable* w.r.t. P if there exists an answer set $(M, \mathcal{H}_M)$ and a labeling function V such that $(\mathcal{H}_M, V : \mathcal{H}_M \rightarrow 2^{\text{Pred}(P)})$, with Pred$(P)$ the predicates in P, is a tree with bounded rank such that

- $p \in V(\varepsilon)$, and
- $p_1 \in V(x)$ iff $p_1(x) \in M$, for a unary predicate p_1 (possibly negated), and
- $p_2 \in V(xi)$ iff $p_2(x, xi) \in M$ or ${p_2}^-(xi, x) \in M$, for a binary predicate p_2 (possibly inverted and/or negated).

A DLP P then has the *tree-model property* if the following property holds: if a unary predicate p (possibly negated) is satisfiable w.r.t. P then p is tree-satisfiable w.r.t. P. For example the predicate *restore* from the example program in the introduction is tree-satisfiable w.r.t. that program, since it has an answer set

$$\{restore(x), crash(x), BackupFailed(x), yest(x, y), yest^-(y, x),$$
$$BackupSucceeded(y), \neg crash(y), yest(y, z), yest^-(z, y),$$
$$BackupSucceeded(z), \neg crash(z), yest(z, u), yest^-(u, z), \ldots\}$$

and this answer set has a tree-structure:



Furthermore, this is the case for every CLP.

Theorem 2. *Every CLP has the tree-model property.*

Proof Sketch. Assume P is a CLP. We can assume that P is such that every X in the head of a rule is the root of the tree representation of that rule, and such that the tree representation is a tree of one level deep [14]. We show that P has the tree-model property, from which we can deduce that every (general) CLP has the tree-model property [14].

Take a unary predicate p (possibly negated) of the CLP P to be satisfiable, i.e. there exists an answer set $(M, \mathcal{H}_M)$ of P such that $p(a) \in M$.

First note that every tree rule in P has a tree representation that is of bounded rank, let m be the maximum rank of the tree representations of all rules, and define n to be the product of m with the number of unary predicates (possibly negated) in P, We define a $\theta : \{1, \ldots, n\}^* \rightarrow \mathcal{H}_M$, such that $(\text{dom}(\theta), t : \text{dom}(\theta) \rightarrow 2^{\text{Pred}(P)})$ is a labeled tree of bounded rank. We define t such that

$$t(xi) = \{p_1 | p_1(\theta(xi)) \in M\} \cup \{p_2 | p_2(\theta(x), \theta(xi)) \in M \vee {p_2}^-(\theta(xi), \theta(x)) \in M\}$$

Define $\theta(\varepsilon) = a$ and assume we have already considered, as in [23], every member of $\{1, \ldots, n\}^k$, as well as $xi1, \ldots, xi(j-1)$ for $xi \in \{1, \ldots, n\}^k$.

For every $p_1 \in t(xi)$, p_1 not free, $xi \in \text{dom}(\theta)$, p_1 unary, we have that $p_1(\theta(xi)) \in M$ and, since M is minimal, there is a rule

$$r_1 : p_1(\theta(xi)) \leftarrow \alpha(\theta(xi)), \gamma_l(\theta(xi), e_l), \epsilon_l(e_l) \ ,$$

such that body$(r_1) \subseteq M$. In the case that $\gamma_l \neq 0$ we proceed as follows.

If there exists a rule[5] (either a tree constraint or a tree rule) with a body

$$\alpha(X), not(\beta(X)), \gamma_1(X, Y_1), \ldots, \gamma_n(X, Y_n), \\ not(\delta_1(X, Y_1)), \ldots, not(\delta_n(X, Y_n)), Y_k \neq Y_l, \epsilon_1(Y_1), \ldots, \epsilon_n(Y_n)$$

such that there exist $n-1$ nodes y_i corresponding to Y_i, with $y_i \in \{xi1, \ldots, xi(j-1), xij, \ldots, xi(j+(l-1))\}$ or $y_i = x$ such that

- $y_i \neq y_j$ if $Y_i \neq Y_j$ in the body,
- $\alpha \subseteq t(xi)$,
- $\beta \cap t(xi) = \emptyset$,
- for all y_i,
  - if $y_i = x$, then $\overline{\gamma_i} \subseteq t(xi)$, $\overline{\delta_i} \cap t(xi) = \emptyset$, and $\epsilon_i \subseteq t(y_i)$, where $\overline{\gamma_i}$ is γ_i with all binary literals inverted,
  - if $y_i \neq x$, then $\gamma_i \subseteq t(y_i)$, $\delta_i \cap t(y_i) = \emptyset$, and $\epsilon_i \subseteq t(y_i)$,
- for the one remaining Y_j, $j \neq i$, we have that $\gamma_j \subseteq \{f | f(\theta(xi), e_l) \in M\}$, $\delta_j \cap \{f | f(\theta(xi), e_l) \in M\} = \emptyset$, and $\epsilon_j \subseteq \{q | q(e_l) \in M\}$

and we have that the body cannot be made true w.r.t. M and $\theta(xi)$ corresponding to X, $\theta(y_i)$, $i \neq j$ corresponding to Y_i and e_l corresponding to Y_j, then $\theta(xi(j+l))$ is undefined, else $\theta(xi(j+l)) = e_l$.

Define $M' = \{p_1(x) | p_1 \in t(x)\} \cup \{p_2(x, xi), p_2{}^-(xi, x) | p_2 \in t(xi)\}$ and $\mathcal{H}_{M'} = \text{dom}(\theta)$. This model clearly makes p tree-satisfiable, if $(M', \mathcal{H}_{M'})$ is an answer set of P, which is the case. □

The decidability proof of satisfiability checking of unary predicates w.r.t. a CLP uses then a reduction to a finite number of simple CLPs for which satisfiability can be checked with normal finite answer set programming. The details can be found in [14].

4 Simulating Description Logics and Finite Answer Set Programming

CLPs can simulate several expressive DLs as well as answer set programming with a finite (Herbrand) universe and without disjunction in the head (i.e. *datalog programs*, where $not()$-literals may appear in the body of a clause). E.g. the program $q(X) \leftarrow f(c, b, c)$ has $\{b, c\}$ as its Herbrand universe. This universe is finite (if it is assumed that a DLP consists of a finite number of rules), contrary to the answer set programming

[5] Note that the tree rules/constraints are trees of one level deep.

introduced in Section 2 where the universe is a superset (possibly infinite) of $\{c, b\}$. However, one can translate $q(X) \leftarrow f(c, b, c)$ into a CLP by grounding it with its Herbrand universe, thus obtaining another finite DLP, and subsequently, a CLP by attaching a variable to it. For the above example, this yields the clauses $q(b)(X) \leftarrow f(c, b, c)(X)$ and $q(c)(X) \leftarrow f(c, b, c)(X)$, with the grounded literals now considered as unary predicates. One obtains the following theorem.

Theorem 3. *M is an answer set of a logic program P iff $(M', \{a\})$, with 'a' a constant, is an answer set of the CLP P' where $M' = \{l(a) | l \in M\}$ 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 $l(X)$.*

Moreover, several DLs that cannot be simulated by finite answer set programming, because they do not have the finite model property, i.e. some DL knowledge bases have only infinite models, can be simulated by CLPs. Such a DL is for example $\mathcal{SHIQ}$ [16], which is the DL that can provide the formal semantics of the ontology language OIL [15], with transitive closure of roles instead of transitivity of roles, which we called $\mathcal{SHIQ}^*$.

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

$$\begin{aligned} D_1, D_2 &\rightarrow A | \neg D_1 | D_1 \sqcap D_2 | D_1 \sqcup D_2 | \exists R.D_1 | \forall R.D_1 | (\leq n\ Q.D_1) | (\geq n\ Q.D_1) \\ Q &\rightarrow P | P^- \\ R &\rightarrow Q | Q^* \end{aligned}$$

with A a concept name and P a role name. The semantics of a $\mathcal{SHIQ}^*$ 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.

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

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 transitive closures). 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 $\mathcal{SHIQ}^*$ 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. $C^{\mathcal{I}} \neq \emptyset$. It is straightforward to simulate satisfiability checking in $\mathcal{SHIQ}^*$ with CLPs.

Consider for example the small fragment of an OWL DL[6] ontology in Figure 1 which expresses that sales items are the items that have at least one price. The DL

```
<owl:Class rdf:ID="SalesItem">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Item"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasPrice"/>
      <owl:minCardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:minCardinality>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Fig. 1. An OWL DL example ontology

knowledge base corresponding to the ontology in Figure 1 consists of the axioms[7]

$$SalesItem \sqsubseteq Item \sqcap \exists hasPrice$$
$$Item \sqcap \exists hasPrice \sqsubseteq SalesItem$$

The corresponding CLP makes $SalesItem$, $Item$, $hasPrice$, and $\neg hasPrice$ free:

$$SalesItem(X) \vee not(SalesItem(X)) \leftarrow$$
$$Item(X) \vee not(Item(X)) \leftarrow$$
$$hasPrice(X,Y) \vee not(hasPrice(X,Y)) \leftarrow$$
$$\neg hasPrice(X,Y) \vee not(\neg hasPrice(X,Y)) \leftarrow$$

and contains rules defining the negation of concept expressions appearing in the knowledge base.[8]

$$\neg SalesItem(X) \leftarrow not(SalesItem(X))$$
$$\neg Item(X) \leftarrow not(Item(X))$$
$$\neg \exists hasPrice(X) \leftarrow not(\exists hasPrice(X))$$
$$\neg(Item \sqcap \exists hasPrice)(X) \leftarrow not((Item \sqcap \exists hasPrice)(X))$$

[6] OWL DL [21] is the most expressive fragment of OWL that corresponds to a DL.

[7] $\exists hasPrice$ corresponds to the concept expression $\exists hasPrice.\top$ where $\top$ is the top concept, i.e. $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ for every interpretation $\mathcal{I}$.

[8] Extending CLP to directly support "true negation" ($\neg$) is possible and would simplify the translation.

as well as rules defining the intersection and the exists restriction $\exists hasPrice$:

$$(Item \sqcap \exists hasPrice)(X) \leftarrow Item(X), \exists hasPrice(X)$$
$$\exists hasPrice(X) \leftarrow hasPrice(X, Y)$$

Finally, we express both DL axioms directly as follows,

$$\leftarrow SalesItem(X), not((Item \sqcap \exists hasPrice)(X))$$
$$\leftarrow not(SalesItem(X)), (Item \sqcap \exists hasPrice)(X)$$

which are the only two rules that are strictly necessary to express the knowledge in the OWL ontology; the other rules simulate the DLs semantics and can be automatically derived.

Formally, we define $\Phi(C, \Sigma)$ to be the CLP, obtained from the $\mathcal{SHIQ}^*$ knowledge base Σ and the concept expression C as follows.

$clos(C, \Sigma)$	$\Phi(C, \Sigma)$	
concepts A	$A(X) \vee not(A(X)) \leftarrow$	(1)
role names P	$P(X, Y) \vee not(P(X, Y)) \leftarrow$	(2)
	$\neg P(X, Y) \vee not(\neg P(X, Y)) \leftarrow$	(3)
expressions D		
$D = \neg E$	$\neg E(X) \leftarrow not(E(X))$	(4)
$D = E \sqcap F$	$E \sqcap F(X) \leftarrow E(X), F(X)$	(5)
$D = E \sqcup F$	$E \sqcup F(X) \leftarrow E(X)$	(6)
	$E \sqcup F(X) \leftarrow F(X)$	(7)
$D = \exists Q.E$	$\exists Q.E(X) \leftarrow Q(X, Y), E(Y)$	(8)
$D = \exists Q^*.E$	$\exists Q^*.E(X) \leftarrow E(X)$	(9)
	$\exists Q^*.E(X) \leftarrow Q(X, Y), \exists Q^*.E(Y)$	(10)
$D = \forall R.E$	$\forall R.E(X) \leftarrow \neg\exists R.\neg E(X)$	(11)
$D = (\leq n\ Q.E)$	$(\leq n\ Q.E)(X) \leftarrow \neg(\geq n+1\ Q.E)(X)$	(12)
$D = (\geq n\ Q.E)$	$(\geq n\ Q.E)(X) \leftarrow Q(X, Y_1), \ldots, Q(X, Y_n),$ $E(Y_1), \ldots, E(Y_n), Y_1 \neq Y_2, \ldots$	(13)
$C_1 \sqsubseteq C_2 \in \Sigma$	$\leftarrow C_1(X), not(C_2(X))$	(14)
$R_1 \sqsubseteq R_2 \in \Sigma$	$\leftarrow R_1(X, Y), not(R_2(X, Y))$	(15)

The *closure* $clos(C, \Sigma)$, appearing in the above table, of a concept expression, C and the $\mathcal{SHIQ}^*$ knowledge base Σ, is defined 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 one of the following
  - $D = \neg D_1$, $D_1 \in clos(C, \Sigma)$
  - $D = D_1 \sqcup D_2$, $\{D_1, D_2\} \subseteq clos(C, \Sigma)$
  - $D = D_1 \sqcap D_2$, $\{D_1, D_2\} \subseteq clos(C, \Sigma)$
  - $D = \exists R.D_1$, $\{R, D_1\} \subseteq clos(C, \Sigma)$
  - $D = \forall R.D_1$, $\{D_1, \exists R.\neg D_1\} \subseteq clos(C, \Sigma)$
  - $D = (\leq n\ Q.D_1)$, then $\{(\geq n+1\ Q.D_1)\} \subseteq clos(C, \Sigma)$
  - $D = (\geq n\ Q.D_1)$, then $\{Q, D_1\} \subseteq clos(C, \Sigma)$
- for all $R^* \in clos(C, \Sigma)$, $R \in clos(C, \Sigma)$,
- for all $D \in clos(C, \Sigma)$, $\neg D \in clos(C, \Sigma)$.

Theorem 4. *A $\mathcal{SHIQ}^*$ concept expression C is satisfiable w.r.t. a $\mathcal{SHIQ}^*$ knowledge base Σ iff $C(X)$ is satisfiable w.r.t. $\Phi(C, \Sigma)$.*

Proof Sketch. $\boxed{\Rightarrow}$ C is satisfiable w.r.t. Σ, so there exists a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $C^{\mathcal{I}} \neq \emptyset$. We construct the answer set $A = (M, \mathcal{H}_M)$ out of this interpretation with $\mathcal{H}_M = \Delta^{\mathcal{I}}$ and M as follows

$$\begin{aligned} M = \{C(a) \,|a \in C^{\mathcal{I}}, C \in \mathit{clos}(C, \Sigma)\} \cup \{\neg C(a) \,|a \notin C^{\mathcal{I}}, C \in \mathit{clos}(C, \Sigma)\} \\ \cup \{Q(a,b), Q^-(b,a) \,|(a,b) \in Q^{\mathcal{I}}, Q \in \mathit{clos}(C, \Sigma)\} \\ \cup \{\neg Q(a,b), \neg Q^-(b,a) \,|(a,b) \notin Q^{\mathcal{I}}, Q \in \mathit{clos}(C, \Sigma)\} \end{aligned}$$

It is then easy to show that $(M, \mathcal{H}_M)$ is an answer set of $\Phi(C, \Sigma)$.

$\boxed{\Leftarrow}$ Let M be a minimal model of $\Phi(C, \Sigma)^M_{\mathcal{H}_M}$ with $C(a) \in M$, and define an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, with $\Delta^{\mathcal{I}} = \mathcal{H}_M$, and $A^{\mathcal{I}} = \{a \,|A(a) \in M\}$, for concept names A, $Q^{\mathcal{I}} = \{(a,b) \,|Q(a,b) \in M\}$, for role names or an inverse Q.

$\mathcal{I}$ is defined on concept expressions as usual, and one can show that $\mathcal{I}$ is a model of Σ such that $C^{\mathcal{I}} \neq \emptyset$. □

Note that while every $\mathcal{SHIQ}^*$ knowledge base can be rewritten, by Theorem 4, as an equivalent CLP, not every CLP can be written as a $\mathcal{SHIQ}^*$ knowledge base expressing the same knowledge. Consider for example the rule

$$g(X, Y) \leftarrow a(X), f(X, Y), b(Y)$$

stating that g is exactly the projection of f on both its first and second coordinate. One direction (the minimality) can be simulated by the three axioms $\top \sqsubseteq \forall g^-.a$, $\top \sqsubseteq \forall g.b$ and $g \sqsubseteq f$. The other direction would demand for a more expressive DL including product of concept expressions and intersection of roles [5].

5 Conclusions and Directions for Further Research

We presented conceptual logic programming (CLP) as a language that unifies both answer set programming and expressive description logics, exemplified by $\mathcal{SHIQ}^*$. This was achieved by, on the one hand, allowing inverse predicates and infinite domains and, on the other hand, suitably restricting the form of clauses so as to keep the satisfiability problem decidable.

Because ontology languages such as OIL, DAML+OIL and a large fragment of OWL, obtain their formal semantics through a correspondence with a description logic, CLP is useful to represent and reason about ontologies in a rule-based manner which also supports fine grained modularity, where ontologies can be extended by simply adding intuitive (business) rules. In addition, reasoning using CLP is nonmonotonic (through negation as failure), an important feature in view of the evolving nature of knowledge that is available on the Semantic Web.

Future work includes extending CLP, e.g. by supporting constants and further relaxing the restrictions on tree rules, possibly even dropping the reliance on the tree model property to guarantee satisfiability. In another direction, CLP could be equipped with

a preference order on rules, thus introducing another source for nonmonotonic reasoning [12], which would be useful for resolving conflicts resulting from the integration of knowledge from different schema's/ontologies. Finally, we intend to confirm the theoretical results with an implementation of CLP.

References

[1] G. Alsaç and C. Baral. Reasoning in Description Logics using Declarative Logic Programming. 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] A. Borgida. On the Relative Expressiveness of Description Logics and predicate logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.

[6] M. Cadoli, L. Palopoli, and M. Lenzerini. Datalog and Description Logics: Expressive Power. In *Proceedings of the Sixth International Workshop on Database Programming Languages (DBPL'97)*, number 1369 in Lecture Notes in Computer Science, pages 281–298. Springer-Verlag, 1998.

[7] 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.

[8] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a Nutshell. In R. Dieng et al., editor, *Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2000.

[9] R. Fikes and D. McGuinness. An Axiomatic Semantics for RDF, RDF-S, and DAML+OIL. http://www.w3.org/TR/daml+oil-axioms, December 2001. W3C Note.

[10] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[11] 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)*, pages 48–57, 2003.

[12] S. Heymans and D. Vermeir. A Defeasible Ontology Language. In Robert Meersman and Zahir Tari et al., editors, *Confederated International Conferences: CoopIS, DOA and ODBASE 2002*, number 2519 in Lecture Notes in Computer Science, pages 1033–1046. Springer, 2002.

[13] S. Heymans and D. Vermeir. Integrating Ontology Languages and Answer set Programming. In *Fourteenth International Workshop on Database and Expert Systems Applications*, pages 584–588, Prague, Czech Republic, September 2003. IEEE Computer Society.

[14] S. Heymans and D. Vermeir. Ontology Reasoning using an Extension of Answer Set Programming. Technical report, Vrije Universiteit Brussel, Dept. of Computer Science, 2003.

[15] I. Horrocks. A Denotational Semantics for Standard OIL and Instance OIL. http://www.ontoknowledge.org/oil/downl/semantics.pdf, 2000.

[16] 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.

[17] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In Harald Ganzinger, David McAllester, and Andrei 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.

[18] J. Kopena. DAMLJessKB. http://plan.mcs.drexel.edu/projects/legorobots/design/software/DAMLJessKB/, October 2002.

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

[20] M. Uschold and M. Grüninger. Ontologies: Principles, Methods, and Applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[21] F. van Harmelen, J. Hendler, I. Horrocks, and L. A. Stein 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.

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

[23] M. Y. Vardi. Reasoning about the Past with Two-Way Automata. In *Proceedings of the 25th Int. Coll. on Automata, Languages and Programming (ICALP '98)*, pages 628–641. Springer, 1998.