# Reasoning in description logics using declarative logic programming

**Güray Alsaç and Chitta Baral**
Department of Computer Sc. and Engg.
Arizona State University
Tempe, AZ, 85287, USA.
{guray,chitta}@asu.edu

**Abstract**

In this paper our goal is to bridge two popular and well-studied knowledge representation formalisms: description logics (DLs), and declarative logic programs (DLPs). In recent years there has been tremendous development in both fields in terms of theoretical studies and implementations. However, despite a few papers on allowing logic programming style rules in description logic, there has been little research on how they relate to each other, how one can be simulated by the other, and how ideas and constructs in one can be used in the other. We show that DLs can be simulated in DLPs and point out why early description logic researchers thought this was not possible. Besides giving a general translation that produces a not so efficient DLP we consider special cases for which more efficient DLPs can be constructed. We also suggest new DL constructs inspired by DLPs.

## Introduction and Motivation

One of the important sub-fields of knowledge representation and reasoning centers around *description logics*, also referred to as *concept languages* and *terminological systems* at different stages of its evolution. Its origin traces back to *frame based systems* and *semantic networks*, both early attempts to represent the classification of objects to a hierarchy of classes (w.r.t. the subset relationship), represent (mostly binary) relationships between classes, and reason with such information. A critical evolutionary step in this field, the KL-ONE system (Brachman & Schmolze 1985), formalized the main ideas in various frame based and semantic network based systems into a logical characterization of classes (or concepts), and relationships (or roles), and proposed a set of constructs to build new classes and relationships from these. Since then, several different description logics have been proposed, each distinguished by the constructs and kinds of relationships allowed. Many of these have been implemented (for example, (Borgida *et al.* 1989)), usually in sound but incomplete fashion. For some, reasoning methodologies have been proposed, and for some the complexity of reasoning has been analyzed. In analyzing the complexity it has been noticed that sometimes seemingly minor syntactic extensions increases the complexity drastically. Recently many attempts have been made in developing expressive description logics with greater applicability.



For example, in (Calvanese, DeGiacomo, & Lenzerini 2001) identification constraints and functional dependencies are added to the description logic $\mathcal{DLR}$, which already includes n-ary relations. In (Haarslev & Moller 2000) a description logic with number restrictions, role hierarchies and transitively closed roles is proposed. In (Levy & Rousset 1996; Cadoli, Palopoli, & Lenzerini 1997; Donini *et al.* 1998) description logics are augmented with Datalog constructs and hybrid languages are proposed.

We now give a small example of representation using description logic and then discuss the applicability and usefulness of description logics. Consider a simple hierarchy of concepts with $person$ at the top (meaning every object in the world is a person) and beneath it the atomic concepts $male$ and $female$. Also, let $childof$ be an atomic relation. Using these we can define a new concept $child$ consisting of elements $x$ such that there exists a person $y$ and $(x, y)$ belongs to the relation $childof$ (i.e., $x$ is a child of $y$). In classical logic this is expressed as $child(x) \equiv \exists y\ childof(x, y) \wedge person(y)$. In description logic it is said that the concept $child$ can be formed using the concept $person$ and the role $childof$ using the construct $\exists^{\geq n}$ as $\exists^{\geq 1} childof.person$. Since $person$ is the top concept, it may be skipped and it is enough to write $\exists^{\geq 1} childof$ or simply $\exists childof$. Similarly, a concept $son$ can be formed by $male \sqcap child$ meaning that the concept $son$ consists of elements who are both $male$ and $child$. In Borgida's (Borgida 1992) syntax, the above definitions of $child$ and $son$ are expressed as $child$ = **at-least**$[1, childof]$ and $son$ = **and**$[male, child]$.

Many consider such description logic expressions to be easier to write and follow than the corresponding expression in classical logic, as in the former the variables are not explicitly specified. Some query languages for querying object oriented databases also have a similar syntax.

A knowledge base in a description logic consists of two parts traditionally referred to as the $TBox$ (meaning "Terminological Box") and the $ABox$ (meaning "Assertional Box"). The first consists of several assertions about concepts and roles, such as the definition of $child$ and $son$ in the above example, and the second consists of specific facts about a particular object belonging to a concept or a particular pair of objects belonging to a particular role, such as Jim being

the child of Mary.

In the absence of an $ABox$, the questions of interest are whether two concepts are equivalent, whether one concept subsumes another, whether a concept is consistent (is non-empty), or whether a knowledge base as a whole is consistent. For example, from a knowledge base consisting of the $TBox$ with the assertion $son$ = **and**$[male, child]$, we can conclude that the concepts $male$ and $child$ subsume the concept $son$. I.e., every son is both a male and a child. At first glance it is often not clear to many people – mainly to those not familiar with description logics – why we should care about subsumption between concepts. This becomes clear when we consider reasoning in the presence of an $ABox$. Given a knowledge base with a non-empty $ABox$, the questions of interest are whether a particular object belongs to a particular concept or whether two objects have a particular role. To answer the first, concept subsumption is often an intermediate step. For example, given that $john$ is a $son$ we can easily conclude that $john$ is a $male$ if we are able to figure out that the concept $male$ subsumes the concept $son$. Besides such reasoning, another important usefulness of description logic is that it can be used in expressing intensional answers to queries to a database. In other words instead of giving an answer as a set of objects, the answer can be expressed as a concept defined using DL constructors in terms of other concepts and classes. Such answers are often more informative than the standard approach of listing individual objects.

Similar to the origin of logic programming, where a subset of first-order formulas called Horn clauses were chosen for efficient reasoning, initial description logics focused on a select set of constructors that were expected to lead to efficient implementations. Many of the early implementations were subsequently found to be incomplete and since then expressiveness has also become an important issue in DL and many new constructors have been proposed.

Currently many people view DL to be useful in expressing *a particular part* of a knowledge base that is about concepts and roles and expect the complete knowledge base to consist of other parts in other knowledge representation languages. In recent years declarative logic programming (DLP), with a critical mass of building block results, available implementations and applications (Baral 2002), has been a leading contender as a knowledge representation language. *In this paper our goal is to show how reasoning in particular DLs can be expressed in DLPs. In particular, we will show how several DL constructs can be expressed in DLP.* In this regard it should be noted that although some of the initial DLs have been shown (Borgida 1992) to be easily translated to subsets of first-order logic, many of the recent and more expressive constructs, such as transitive closure of roles, are not amenable to a first-order translation.

The motivation behind our goal to formulate DL reasoning as reasoning in a DLP translation is manyfold: (i) It may lead to easier integration of DL features into DLP knowledge bases. (ii) It will help both the DL community and the DLP community to understand the advantages of the other approach. In particular, it will help clear misunderstandings in the DL community about the expressiveness of DLP. Although the DL community has looked into DLP features, they (Donini *et al*. 1998; Levy & Rousset 1996; Cadoli, Palopoli, & Lenzerini 1997) have mostly concentrated on Datalog augmented with stratified negation. The expressiveness of DLP is vastly increased by allowing unrestricted negation as failure characterized by the answer set (stable model) semantics, and by using disjunctions and function symbols. (iii) The translation of DL constructs to DLP will also shed light on some strong points of DL and how to encode them in a DLP. (iv) This may lead to identifying parts of a DLP that specify knowledge that can be otherwise expressed and reasoned efficiently using DLs.

The rest of this paper is organized as follows. We first formally define a DL and its semantics and also DLP and its semantics. We then present a general translation from a description logic to DLP and then present a translation for a sub-class that generates a more efficient DLP. Subsequently we briefly mention how some of the constructs in other description logics can be expressed in DLP, suggest some future DL constructs that are inspired by DLP and conclude.

# Background

## Description Logics

In this section we define the description logic language $\mathcal{ALCQI}$ from the recent paper (Calvanese *et al.* 2001).

**Concepts and Roles: syntax and semantics** Concepts and roles in $\mathcal{ALCQI}$ are formed using the following syntax:

$$C, C' \longrightarrow A \mid \neg C \mid C \sqcap C' \mid C \sqcup C' \mid \forall R.C \mid \exists R.C \mid \exists^{\geq n} R.C \mid \exists^{\leq n} R.C$$

$$R \longrightarrow P \mid P^-$$

where $A$ and $P$ are atomic concepts and atomic roles respectively, $C$ and $R$ denote arbitrary concepts and roles and $n$ denotes a positive integer. For readability, for any atomic concept $A$, $A \sqcap \neg A$ is abbreviated as $\bot$, and $A \sqcup \neg A$ is abbreviated as $\top$. Similarly, $\neg C \sqcup D$ is abbreviated as $C \Rightarrow D$, $\exists^{\geq n} R.C \sqcap \exists^{\leq n} R.C$ is abbreviated as $\exists^{=n} R.C$, and $R.\top$ is abbreviated as $R$.

Concepts are interpreted as subsets of a domain, and roles as binary relations over that domain. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over a set $\mathcal{A}$ of atomic concepts and a set $\mathcal{P}$ of atomic roles consists of a nonempty set $\Delta^{\mathcal{I}}$ (the domain of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ that maps every atomic concept $A \in \mathcal{A}$ to a subset of $\Delta^{\mathcal{I}}$ and every atomic role $P \in \mathcal{P}$ to a subset $P\mathcal{I}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to arbitrary concepts and roles as follows:

$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$.
$(C \sqcap C')^{\mathcal{I}} = C^{\mathcal{I}} \cap C'^{\mathcal{I}}$. $\quad (C \sqcup C')^{\mathcal{I}} = C^{\mathcal{I}} \cup C'^{\mathcal{I}}$.
$(\forall R.C)^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} \mid \forall o'.(o, o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$.
$(\exists R.C)^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} \mid \exists o'.(o, o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\}$.
$(\exists^{\geq n} R.C)^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} : |\{o' \mid (o, o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\}| \geq n\}$.
$(\exists^{\leq n} R.C)^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} : |\{o' \mid (o, o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\}| \leq n\}$.
$(R^-)^{\mathcal{I}} = \{(o, o') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (o', o) \in R^{\mathcal{I}}\}$.

**Definition 1** A concept $C$ is said to be *satisfiable* if it has a non-empty interpretation.

Given two concepts $C_1$ and $C_2$, we say $C_2$ *subsumes* $C_1$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ holds in every interpretation. □

**Knowledge bases: TBox and ABox** A knowledge base (in $\mathcal{ALCQI}$) consists of two parts: a $TBox$ and an $ABox$. The $TBox$ consists of a finite set of *inclusion assertions* of the form $C_1 \sqsubseteq C_2$.

where $C_1$ and $C_2$ are concepts. Often we will abbreviate the two assertions $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$ by the single statement $C_1 \equiv C_2$.

If $C_1$ above is an atomic concept then it is referred to as a *primitive inclusion assertion* and statements of the form $C_1 \equiv C_2$ are then referred to as *equality assertions*.

An interpretation $\mathcal{I}$ is said to satisfy an assertion of the form $C_1 \sqsubseteq C_2$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.

The $ABox$ consists of *fact assertions* of the form: $C(a)$ and $R(a, b)$

where $C$ is a concept, $R$ is a role, and $a, b$ are elements of a new alphabet $HB$. In presence of non-empty $ABox$ the interpretation function $\cdot^{\mathcal{I}}$ is extended to individuals in $HB$ such that $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each individual $a \in HB$ and $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$.

A fact assertion $C(a)$ is said to be satisfied by an interpretation $\mathcal{I}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and a fact assertion $R(a, b)$ is said to be satisfied by an interpretation $\mathcal{I}$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in C^{\mathcal{I}}$.

**Definition 2** An interpretation is said to be a model of a knowledge base $\mathcal{K}$ if it satisfies all assertions in it; $\mathcal{K}$ is said to be *satisfiable* if it has a model; a concept $C$ is said to be *consistent* in $\mathcal{K}$ if $\mathcal{K}$ has a model $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$; and for an assertion $\alpha$, we say $\mathcal{K} \models \alpha$, if $\alpha$ is satisfied by each model $\mathcal{I}$ of $\mathcal{K}$. □

Since in logic programming the semantics is defined with respect to a Herbrand Universe, we will define a restricted notion of interpretation, models and entailment for DLs. An HB-interpretation is an interpretation where the domain is $HB$. An HB-interpretation is said to be an HB-model of a knowledge base $\mathcal{K}$ if it satisfies all assertions in it; and $\mathcal{K} \models_{HB} \alpha$, if $\alpha$ is satisfied by each HB-model $\mathcal{I}$ of $\mathcal{K}$.

## Declarative Logic Programming with answer sets

A declarative logic program (DLP) is a collection of rules of the form

$$l_0 \textit{ or } \ldots \textit{ or } l_k \leftarrow l_{k+1}, \ldots, l_m, \mathbf{not}\ l_{m+1}, \ldots, \mathbf{not}\ l_n \quad (1)$$

where $l_i$'s are literals in the sense of classical logic.[1] For a literal $l$, "$\mathbf{not}\ l$" is referred to as a naf-literal. Intuitively, the above rule means that if $l_{k+1} \ldots l_m$ are believed to be true and $l_{m+1} \ldots l_n$ can be assumed to be false then at least one of $l_0, \ldots, l_k$ must be believed as true. In the notation of (Baral 2002) a DLP of the above form is referred to as an AnsProlog $^{or, \neg}$ program. If $k = 0$ (i.e., there is no disjunction) and if $l_i$'s are atoms (i.e., there is no classical negation) then the program consisting of such rules is referred to as an AnsProlog program (also called a normal or general logic program), and furthermore if $m = n$, then we have an AnsProlog$^{-\mathbf{not}}$ program (also called a definite program).

The semantics of arbitrary AnsProlog $^{or, \neg}$ programs are with respect to the grounding of the program whereby the variables in the rules are substituted by elements of the Herbrand universe consisting of all the ground terms made up of constants, and function symbols in the language. We now present the semantics of AnsProlog programs (which we will use in this paper) and refer to (Baral 2002) and the DLP literature for the semantics of AnsProlog $^{or, \neg}$ programs. In our definition we only consider ground AnsProlog programs.

AnsProlog$^{-\mathbf{not}}$ programs – also referred to as definite programs – have unique answer sets, which are the least models of the theory obtained by treating rules of the form $a_0 \leftarrow a_1, \ldots, a_m$ (where $a_i$s are ground atoms) as the classical formula $a_1 \wedge \ldots \wedge a_m \supset a_0$. Given an AnsProlog program $P$ and a set of atoms $S$, the Gelfond-Lifschitz transformation $P^S$ is defined as the set of rules obtained from $P$ by removing all rules from $P$ whose body contains $\mathbf{not}\ b$ such that $b \in S$, and then removing the naf-literals from the rest of the rules.

**Definition 3** A set $S$ of atoms is said to be an answer set (also referred to as stable model) of an AnsProlog program $P$ if $S$ is the answer set of the definite program $P^S$.

Given an AnsProlog program $P$ and a ground atom $a$, we say $P$ entails $a$, denoted by $P \models_{dlp} a$, if $a$ is true in all answer sets of $P$. □

Notable among the major recent developments on DLP are: (i) development of several DLP interpreters, most notably smodels(Niemela & Simons 1997) and dlv (Citrigno *et al.* 1997); (ii) various results on how to represent knowledge, reason and declaratively express problem solving tasks, and systematically build DLP programs as compiled in (Baral 2002); (iii) complexity and expressibility results of various DLP classes as compiled in (Dantsin *et al.* 1999); and (iv) use of DLP in various applications such as planning, product configuration, cryptography etc.

Since in this paper we are comparing two different knowledge represent languages we would like to briefly mention some of the complexity and expressiveness results about AnsProlog $^{or}$ and its sub-classes.

- function-free programs: Here the complexity results are divided into two kinds: program complexity and data complexity. For AnsProlog program the program and data complexity is co-NEXPTIME complete and co-NP complete respectively; and for AnsProlog $^{or}$ programs the program and data complexity is co-NEXPTIME$^{NP}$ complete and $\Pi_2$P complete respectively.

  In terms of expressibility, AnsProlog captures the class co-NP while AnsProlog $^{or}$ captures the class $\Pi_2$P.

[1] Although in our translation from DL to DLP we will not use programs with *or* , for giving a big picture of DLP to people not familiar with them we consider them here. Also, we will allow rules with empty heads as they can be compiled away. For example: the rule $\leftarrow$ *body* can be replaced by the rule $p \leftarrow \mathbf{not}\ p, body$,where $p$ is a new atom.

- programs with functions: The complexity of both AnsProlog and AnsProlog$^{or}$ programs in presence of functions is $\Pi^1_1$ and they both capture the class $\Pi^1_1$.

## Translating $\mathcal{ALCQI}$ to DLP

Before we give a general translation from $\mathcal{ALCQI}$ to DLP we will first illustrate the problem with a straight-forward and simplistic translation. (This translation was used in (Borgida 1992) to demonstrate an inadequacy of logic programming in modeling such knowledge.) A straightforward translation of the TBox assertions (from Section ) $child = \exists\, childof$ and $son = male \sqcap child$ to declarative logic programming would be the following rules.

$r_1 : child(X) \leftarrow childof(X, Y).$
$r_2 : son(X) \leftarrow male(X), child(X).$

The above DLP is equivalent to the first-order formula $(\forall X.child(X) \Longleftrightarrow \exists Y childof(X, Y)) \wedge$ $(\forall X.son(X) \Longleftrightarrow male(X) \wedge child(X))$. The TBox assertions are also equivalent to the above formulas. Nevertheless, if we add the fact assertion $son(a)$ to the DL knowledge base then we will be able to infer $child(a)$, while the same is not true for the logic program translation. I.e., while $\{child = \exists\, childof, son = male \sqcap child, son(a)\} \models child(a)$, $\{r_1, r_2, son(a)\} \not\models_{dlp} child(a)$.

The above illustrated inadequacy is now well-understood in logic programming and proposals have been made to overcome it. One proposal is to use abductive logic programming (Kakas & Mancarella 1990) and have the predicates $male$ and $childof$ as abducibles and treat $son(a)$ as an observation. An alternative approach is to use non-stratified negation in enumerating the predicates $male$ and $childof$ and assimilate $son(a)$ through filtering (Baral 2002). If we follow this technique, besides $r_1$ and $r_2$, the translation will consist of the following rules:

$r_3 : male(X) \leftarrow top(X), \mathbf{not}\ female(X).$
$r_4 : female(X) \leftarrow top(X), \mathbf{not}\ male(X).$
$r_5 : childof(X, Y) \leftarrow top(X), top(Y), X \neq Y,$
$\qquad \mathbf{not}\ not\_childof(X, Y).$
$r_6 : not\_childof(X, Y) \leftarrow top(X), top(Y), X \neq Y,$
$\qquad \mathbf{not}\ childof(X, Y).$
$r_7 : \leftarrow childof(X, Y), childof(Y, X).$

and facts about the predicate $top$ listing all elements belonging to the concept $\top$.

Now to assimilate the $son(a)$ we need to add the following:

$r_8 : \leftarrow \mathbf{not}\ son(a).$

whose effect is to eliminate all potential answer sets which do not contain $son(a)$.

Suppose the concept $\top$ consists of objects $a$ and $b$. In that case the program $P_1 = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, top(a), top(b)\}$ will have twelve answer sets, two of which will be answer sets of the program $P_2$ obtained by adding $r_8$ to $P_1$. These two are $\{top(a), top(b), male(a), male(b), childof(a, b), not\_childof(b, a), child(a), son(a)\}$ and $\{top(a), top(b), male(a), female(b), childof(a, b), not\_childof(b, a), child(a), son(a)\}$. Since $child(a)$ belongs to both of them we have $P_2 \models_{dlp} child(a)$.

We now give a general translation from $\mathcal{ALCQI}$ to DLP, first focusing only on entailment of fact assertions.

### A general DLP translation for entailing fact assertions

In the following if capital letter $C$ is a concept and capital letter $R$ is a role then the small letter $c$ is a predicate that corresponds to $C$ and the small letter $r$ is a predicate that corresponds to $R$.

**Step 1**: For each element $a$ of $HB$ the translation contains the fact: $top(a) \leftarrow.$

**Step 2**: For each atomic concept $B$ the translation will have the rules:

$b(X) \leftarrow top(X), \mathbf{not}\ not\_b(X).$
$not\_b(X) \leftarrow top(X), \mathbf{not}\ b(X).$

**Step 3**: For each atomic role $P$ the translation will have the rules:

$p(X, Y) \leftarrow top(X), top(Y), \mathbf{not}\ not\_p(X, Y).$
$not\_p(X, Y) \leftarrow top(X), top(Y), \mathbf{not}\ p(X, Y).$

**Step 4**: For each fact assertion of the form $C(a)$, the translation will have $tr(C)$ as described in Step 7 and the following: $\leftarrow \mathbf{not}\ c(a).$

**Step 5**: For each fact assertion of the form $R(a, b)$, the translation will have $tr(R)$ as described in Step 8 and the following: $\leftarrow \mathbf{not}\ r(a, b).$

**Step 6**: For each inclusion assertion of the form $C_1 \sqsubseteq C_2$, the translation will have rules defining predicates $c_1$ and $c_2$ as described in Step 7 below and the following (if $C_1$ and $C_2$ are non atomic concepts then $c_1$ and $c_2$ will be new predicates): $\leftarrow c_1(X), \mathbf{not}\ c_2(X).$

**Step 7**: For a concept expression $C$, its definition in terms of a predicate $c$ denoted by $tr(C)$ is as follows:

1. If $C$ is of the form $\neg C'$, then $tr(C)$ contains the rule
   $c(X) \leftarrow top(X), \mathbf{not}\ c'(X).$
   and the translation $tr(C')$.
2. If $C$ is of the form $C_1 \sqcap C_2$, then $tr(C)$ contains the rule
   $c(X) \leftarrow top(X), c_1(X), c_2(X).$
   and the translations $tr(C_1)$, and $tr(C_2)$.
3. If $C$ is of the form $C_1 \sqcup C_2$, then $tr(C)$ contains the rules
   $c(X) \leftarrow top(X), c_1(X).$
   $c(X) \leftarrow top(X), c_2(X).$
   and the translations $tr(C_1)$, and $tr(C_2)$.
4. If $C$ is of the form $\forall R.C'$, then $tr(C)$ contains the rules
   $not\_c(X) \leftarrow r(X, Y), \mathbf{not}\ c'(Y).$
   $c(X) \leftarrow top(X), \mathbf{not}\ not\_c(X).$
   and the translations $tr(R)$ (to be defined in Step 8), and $tr(C')$.
5. If $C$ is of the form $\exists R.C'$, then $tr(C)$ contains the rule
   $c(X) \leftarrow r(X, Y), c'(Y).$
   and the translations $tr(R)$ (defined in Step 8), and $tr(C')$.
6. If $C$ is of the form $\exists R^{\geq n}.C'$, then $tr(C)$ contains the rule
   $c(X) \leftarrow r(X, Y_1), \ldots, r(X, Y_n), c'(Y_1), \ldots, c'(Y_n),$
   $\qquad Y_1 \neq Y_2 \neq \ldots \neq Y_n.$
   and the translations $tr(R)$ (defined in Step 8), and $tr(C')$.

7. If $C$ is of the form $\exists R^{\leq n}.C'$, then $tr(C)$ contains the rule
$not\_c(X) \leftarrow r(X,Y_1),\ldots,r(X,Y_{n+1}),c'(Y_1),\ldots,c'(Y_{n+1}),$
$Y_1 \neq Y_2 \neq \ldots \neq Y_{n+1}.$
$c(X) \leftarrow top(X), \mathbf{not}\ not\_c(X).$
and the translations $tr(R)$ (to be defined in Step 8), and $tr(C')$.

**Step 8**: For an arbitrary role $R$ its definition in terms of a predicate $r$ denoted by $tr(R)$ is as follows:

1. If $R$ is an atomic role $P$, then $tr(R)$ contains the rule $r(X,Y) \leftarrow p(X,Y)$.
2. If $R$ is of the form $P^-$, where $P$ is an atomic role, then $tr(R)$ contains the rule $r(X,Y) \leftarrow p(Y,X)$.

Given a DL knowledge base $\mathcal{K}$, we will denote the DLP obtained by the above steps by $tr(\mathcal{K})$.

**Example 1** Lets us now consider the example in the beginning of this section and translate it into DLP using $tr$. The DL knowledge base consists of $\{child \equiv \exists childof.top, son \equiv male \sqcap child, son(a)\}$, with $HB = \{a,b\}$.

- Step 1. $top(a) \leftarrow.$ $top(b) \leftarrow.$
- Step 2. $child(X) \leftarrow top(X), \mathbf{not}\ not\_child(X).$
$not\_child(X) \leftarrow top(X), \mathbf{not}\ child(X).$
$male(X) \leftarrow top(X), \mathbf{not}\ not\_male(X).$
$not\_male(X) \leftarrow top(X), \mathbf{not}\ male(X).$
$son(X) \leftarrow top(X), \mathbf{not}\ not\_son(X).$
$not\_son(X) \leftarrow top(X), \mathbf{not}\ son(X).$
- Step 3.
$childof(X,Y) \leftarrow top(X), top(Y), \mathbf{not}\ not\_childof(X,Y).$
$not\_childof(X,Y) \leftarrow$
$top(X), top(Y), \mathbf{not}\ childof(X,Y).$
- Step 4. $\leftarrow \mathbf{not}\ son(a).$
- Step 6.
$\leftarrow child(X), \mathbf{not}\ c(X).$ $\leftarrow c(X), \mathbf{not}\ child(X).$
$\leftarrow son(X), \mathbf{not}\ c'(X).$ $\leftarrow c'(X), \mathbf{not}\ son(X).$
- Step 7. $c(X) \leftarrow childof(X,Y), top(Y).$
$c'(X) \leftarrow male(X), child(X).$ □

**Theorem 0.1** Let $\mathcal{K}$ be a DL knowledge base.
$\mathcal{K} \models_{HB} C(a)$ if and only $tr(\mathcal{K}) \models_{dlp} c(a)$ and
$\mathcal{K} \models_{HB} R(a,b)$ if and only if $tr(\mathcal{K}) \models_{dlp} r(a,b)$.[2] □

## Entailing inclusion assertions and other reasoning tasks

In this section we discuss a translation with respect to $\models$ (instead of $\models_{HB}$) and consider reasoning tasks such as if a DL knowledge base $\mathcal{K}$ entails a inclusion assertion; or if it is satisfiable; or if a concept is consistent in it. We now give the translation $tr'$.

[2]Note that this theorem is not true for $\models$ instead of $\models_{HB}$. The discrepancy may arise when $C$ is a concept using universal quantifiers. For example, as suggested by an anonymous reviewer, if $K = \{C(a), R(a,a)\}$ with $HB = \{a\}$ then $K \not\models \forall R.C(a)$, while $K \models_{HB} \forall R.C(a)$. In logic programming, this discrepancy is referred to as the universal query problem and several ways to overcome it are proposed in the literature. In the next section we use one such proposal.

**Step 1'**: For each element $a$ of $HB$ the translation contains the fact: $top(a) \leftarrow.$
In addition, the translation[3] contains the following:
$top(a') \leftarrow.$ $top(f(X)) \leftarrow top(X).$
where $a'$ is an object constant that does not appear in $\mathcal{K}$, and $f$ is a unary function symbol used to construct an infinite Herbrand Universe.

**Step 2-8** remain the same as in Section . Given a DL knowledge base $\mathcal{K}$, we will denote the DLP obtained by the above steps by $tr'(\mathcal{K})$.

**Step 9**: For a query $\alpha$ given as $C_1 \sqsubseteq C_2$, the translation $qtr(\alpha)$ will have (a) rules defining two new predicates $c_1$ and $c_2$ as described in Step 7 and (b) the following.
$violated \leftarrow c_1(X), \mathbf{not}\ c_2(X).$
$satisfied \leftarrow \mathbf{not}\ violated.$

**Theorem 0.2** Let $\mathcal{K}$ be a DL knowledge base and $\alpha$ be an inclusion assertion.
$\mathcal{K} \models \alpha$ if and only if $tr'(\mathcal{K}) \cup qtr(\alpha) \models_{dlp} satisfied.$ □

**Lemma 0.3** Let $\mathcal{K}$ be a DL knowledge base. $C_2$ subsumes $C_1$ in $\mathcal{K}$ iff $\mathcal{K} \models C_1 \sqsubseteq C_2$. □

**Theorem 0.4** Let $\mathcal{K}$ be a DL knowledge base.
(i) $\mathcal{K}$ is satisfiable iff $tr'(\mathcal{K})$ has an answer set.
(ii) A concept $C$ is consistent in $\mathcal{K}$ iff $tr'(\mathcal{K}) \cup \{non\_empty\_c \leftarrow c(X), top(X)\}$ has an answer set containing $non\_empty\_c$. □

## A translation for DL knowledge bases with non-cyclic equality assertions

The translation in the previous two sections are general in the sense that they do not restrict the inclusion assertions. The program obtained by the translation enumerates all the atomic concepts and thus may result in a large search space while computing the answer sets. In this section we consider a restriction on the inclusion assertions and present a translation that cuts down on the enumeration and hence reduces the search space during answer set computation.

The restriction that we impose, which is also imposed in many early DL systems, is that the TBox consist only of equality assertions with an atomic concept in the left hand side and that there be no cycles in the sense that no concept in the right hand side of an equality assertion refers (either directly or indirectly) to the atomic concept in the left part of the assertion.

Given such a TBox we refer to those atomic concepts that appear in the left hand side of an equality assertion as *derived atomic concepts* and all other atomic concepts as *non-derived atomic concepts*. We now describe the translation $tr_n$.

The translation $tr_n$ has the same Step 1, 3, 4, 5, 7 and 8 as $tr$. The Step 2 of $tr_n$ enumerates only the *non-derived atomic concepts* other than $top$. In Step 6, for each equality assertion of the form $A \equiv C$, the translation has $tr(C)$ as described in Step 7 of $tr$ and the rule: $a(X) \leftarrow c(X).$

[3]In cases where we are only concerned with finite domains the second rule can be omitted, making the translation simpler.

**Example 2** Let us now consider our running example with the TBox = $\{child \equiv \exists childof.top; son \equiv male \sqcap child\}$ and ABox = $\{son(a)\}$. With respect to this TBox, $son$ and $child$ are derived atomic concepts while $male$ and $top$ are non-derived atomic concepts. We now show that the program obtained by $tr_n$ for this DL knowledge base is very close to the program $\{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, top(a), top(b)\}$ analyzed in the beginning of Section .

Step 1 of $tr_n$ gives us $top(a)$ and $top(b)$. Step 2 of $tr_n$ gives us $r_3$ and $r_4$ where we enumerate the non-derived concept $male$. Step 3 gives us $r_5$ and $r_6$. Step 4 gives us $r_8$. Step 6 gives us $r_2$ and a slight variant of $r_1$. □

**Theorem 0.5** Let $\mathcal{K}$ be a DL knowledge base whose TBox consists of equality assertions and is non-cyclic.

$\mathcal{K} \models_{HB} C(a)$ if and only $tr_n(\mathcal{K}) \models_{dlp} c(a)$ and
$\mathcal{K} \models_{HB} R(a, b)$ if and only if $tr_n(\mathcal{K}) \models_{dlp} r(a, b)$. □

If it is given that $\mathcal{K}$ is consistent and we are interested in finding out $\mathcal{K} \models_{HB} C(a)$ where $C$ is a derived atomic concept then we can even avoid enumeration of non-derived atomic concepts. We will explore more along these lines in the future.

## Conclusion and Future work

In this paper so far we only focused on the constructs in the DL language $\mathcal{ALCQI}$ and their translation to DLP. In the full paper we will consider translations of a more exhaustive list of DL constructs including concept constructors such as **subset[**$R, P$**]**, **same-as[**$R, P$**]**, **not-same-as[**$R, P$**]**, **fills[**$R, i$**]** and **one-of[**$i_1$**,...,** $i_n$**]** and role constructors such as **role-and[**$R, P$**]**, **role-or[**$R, P$**]**, **role-not[**$R$**]**, **restrict[**$R, A$**]**, **compose[**$R, P$**]**, **product[**$C_1, C_2$**]**, and **trans[**$R$**]**.

Among the above, transitive closure is a construct which is not expressible in first-order logic, but is easily expressible in DLP. For example, if $R$ is the transitive closure of a role $P$, then $tr(R)$ will contain the rules
$r(X, Y) \leftarrow p(X, Y)$.
$r(X, Y) \leftarrow p(X, Z), p(Z, Y)$.

Besides the formal results about expressiveness of AnsProlog $^{or}$ subclasses that we mentioned in Section , its practical implication is that it allows us to (i) enumerate predicates, (ii) perform aggregation, (iii) capture default reasoning, and (iv) do declarative problem solving. We notice that the existing few DL+DLP synthesis papers (Levy & Rousset 1996; Cadoli, Palopoli, & Lenzerini 1997; Donini *et al.* 1998; Toman & Weddell 2001) usually focus on less expressive DLP subclasses. Allowing non-stratification in DLPs by using the answer set semantics enhances the expressive power of DLP and allows us to do (i), (ii) and (iv) which are not easily doable without these constructs (in the absence of function symbols). For example, in (Cadoli, Palopoli, & Lenzerini 1997) uses inclusion assertions to express enumeration and Datalog rules to define a concept while encoding 3-colorability of a graph. AnsProlog can express both aspects of that example.

Although our focus so far was on translating DL to DLP, we now suggest a role constructor inspired by the notion of aggregation in DLPs. Suppose we want to state a new relationship $R^{same}$ using a relationship $R$ with the intuitive meaning that $a$ is related to $b$ by $R^{same}$ if the number of tuples in $R$ with $a$ in its first position is same as the number of tuples in $R$ with $b$ in its first position.

$(R^{same})^{\mathcal{I}} = \{(a, b) \mid card\{o' \mid (a, o') \in R^{\mathcal{I}}\} = card\{o' \mid (b, o') \in R^{\mathcal{I}}\}\}$

The above can not be expressed in logics, such as first order logic, where we can not do aggregation. It can be expressed (Zaniolo, Arni, & Ong 1993) in AnsProlog.

## References


Baral, C. 2002. *Knowledge representation, reasoning and declarative problem solving.* Cambridge University Press. (in press)

Borgida, A.; Brachman, R.; McGuinness, D.; and Resnick, L. 1989. CLASSIC: A structural data model for objects. In *Proc. SIGMOD 89*, 58–67.

Borgida, A. 1992. Description logics are not just for the flightless-birds: A new look at the utility and foundations of description logics. Technical report, Rutgers University.

Brachman, R., and Schmolze, J. 1985. An overview of the KLONE knowledge rep. system. *Cog. Science* 9(2):171–216.

Cadoli, M.; Palopoli, L.; and Lenzerini, M. 1997. Datalog and DLs: expressive power. In *Proc. of DBPL'97*, 281–298.

Calvanese, D.; DeGiacomo, G.; Nardi, D.; and Lenzerini, M. 2001. Reasoning in expressive DLs. In Robinson, A., and Voronkov, A., eds., *Handbook of automated reasoning*. N. Holland.

Calvanese, D.; DeGiacomo, G.; and Lenzerini, M. 2001. Identification constraints and functional dependencies in description logics. In *Proc. of IJCAI 2001*, 155–161.

Citrigno, S.; Eiter, T.; Faber, W.; Gottlob, G.; Koch, C.; Leone, N.; Mateis, C.; Pfeifer, G.; and Scarcello, F. 1997. The dlv system: Model generator and application front ends. In *Proceedings of the 12th Workshop on Logic Programming*, 128–137.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 1999. Complexity and expressive power of logic programming. Technical Report INFSYS 1843-99-05, Technische Universitat Wien.

Donini, F.; Lenzerini, M.; Nardi, D.; and Schaerf, A. 1998. AL-log: Integrating datalog and DLs. *JIIS* 10(3):227–252.

Haarslev, V., and Moller, R. 2000. Expressive abox reasoning with number restrictions, role hierarchies, and transitively closed roles. In *KR'00*, 273–284.

Kakas, A., and Mancarella, P. 1990. Generalized stable models: a semantics for abduction. In *Proc. of ECAI-90*, 385–391.

Levy, A., and Rousset, M. 1996. CARIN: A representation language combining Horn rules and description logics. In *Proc. of ECAI-96*, 323–327.

Niemela, I., and Simons, P. 1997. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In Dix, J.; Furbach, U.; and Nerode, A., eds., *Proc. 4th international conference on Logic programming and non-monotonic reasoning*, 420–429. Springer.

Toman, D., and Weddell, G. 2001. On attributes, roles and dependencies in description logics and the Ackermann case of the decision problem. In *Proc. of DL'01*, 76–85.

Zaniolo, C.; Arni, N.; and Ong, K. 1993. Negation and aggregates in recursive rules: the LDL++ approach. In *DOOD*, 204–221.