Enhancing Disjunctive Logic Programming for Ontology Specification

F. Calimeri[2], S. Galizia[2], M. Ruffolo[1,3], and P. Rullo[1,2]

[1]Exeura s.r.l. Via P. Bucci 87030 Rende (CS), Italy
E-Mail:{ruffolo,rullo}@exeura.it

[2]Universitå della Calabria - Dip. Di Matematica
Via P. Bucci, 87030 Rende (CS), Italy
E-Mail:{calimeri,galizia,rullo}@mat.unical.it

[3]ICAR-CNR, Via P. Bucci, 87030 Rende (CS), Italy
E-Mail:ruffolo@icar.cnr.it

Abstract. This paper is a presentation of ONTO-DLP, an extension of Disjunctive Logic Programming (DLP) for complex knowledge modelling. ONTO-DLP enhances DLP by constructs coming from other knowledge representation formalisms, namely, Complex-Datalog and Ordered Logic. The former provides suitable abstraction mechanisms for complex objects handling, while the latter provides support for default reasoning. Such features make ONTO-DLV a powerful language for ontology specification and reasoning.

1 Introduction

Disjunctive Logic Programming (DLP) – an extension of Datalog where disjunction is allowed in the rules' heads – is nowadays widely recognized as a valuable tool for knowledge representation and reasoning [4, 5]. An important merit of DLP over normal (i.e., disjunction-free) logic programming is its capability to model incomplete knowledge [2, 12]. The presence of disjunction in the heads of the rules makes DLP inherently non-monotonic, that is, new information can invalidate previous conclusions. Much research work has been done on the semantics of DLP and several alternative semantics have been proposed (see [5–7] for comprehensive surveys). The most widely accepted semantics is the extension of the stable model semantics of Gelfond-Lifschitz [6] to disjunctive deductive databases [6, 7]. Stable model semantics for DLP has a very high expressive power: DLP under stable model semantics captures the complexity class $\Sigma_2^P = NP^{NP}$ (i.e., they allow to express every property which is decidable in non-deterministic polynomial time with an oracle in NP). As shown in [8, 3], the high expressive power has also a practical relevance, as concrete real world situations can be represented by DLP, while they cannot be expressed by disjunction-free programs. An interesting extension of DLP by strong and weak constraints has been proposed in [1, 5], and others are currently object of research. A number of large scale systems, capable of handling thousands of rules, have been implemented in the last few years: DLV developed by Leone's group [5, 12], GnT/SModels developed by Niemelä and Simons [13], XSB developed by Warren et al. [14], and DeReS developed by Marek and Truszczynski [2]. While XSB and DeReS support only fragments (i.e., sublanguages) of DLP, DLV and GnT/SModels support full DLP (in particular, DLP supports also strong and weak constraints [1]). The high expressiveness of DLP suggests that these systems have high potential for exploitation in the area of knowledge management. Unfortunately, DLP as such lacks of suitable abstraction mechanisms needed for the representation and the manipulation of complex application domains.

Complex-Datalog [9] is an extension of Datalog which includes a number of object-oriented constructs, namely, object identity, complex schemes, multiple inheritance. The most salient feature of the language is the way it combines the computational power of Datalog with the structural complexity of nested relation and object data models.

Ordered Logic [10, 11] is a non-monotonic reasoning formalism providing support for default reasoning. The basic mechanism of Ordered Logic (OL for short) comes from the combination of true (or classical) negation (i.e., the possibility of explicitly stating the falsity of atoms) and inheritance hierarchies (which assign different levels of reliability to rules). OL can be regarded as an extension of Datalog to deal with true negation and defeasible reasoning. An OL program is a set of components organized into an hierarchical structure. Each component consists of a set of rules which may have negative heads. Like in the object-oriented approach, properties (rules) defined for the "higher" components in the hierarchy flow down to the "lower" ones. Thus, contradicting conclusions may be drawn. A stable model semantics for OL programs has been proposed in [11].

In this paper we present a description of ONTO-DLP, a formalism for ontology specification which combines the full computational power of DLP with the knowledge modeling features of Complex-Datalog and the default reasoning mechanisms of Ordered Logic. ONTO-DLP provides the basic notions of concept, relation and axiom; thanks to the presence of features like disjunction, negation by failure, classical negation, constraints (both, strong and weak) and (multiple) inheritance, it is a formalism capable of supporting very powerful reasoning tasks. The semantics of ONTO-DLP ontologies relies on the notion of stable model and uses a three-value logic where each fact can be either true, false or undefined (maybe fact); as we will se, the proposed semantics is very intuitive for the purpose of modeling inheritance-based reasoning tasks. ONTO-DLP is currently being implemented as a front-end of the disjunctive logic programming system DLV [5, 12].

2 The Living Being Ontology (a simple hierarchy)

The first example shows an ontology consisting of a simple hierarchy of living beings.

```
LIVING_BEING
    { mortal() ← }
      ANIMAL isa LIVING_BEING
        MAMMAL isa ANIMAL
          CARNIVORE  isa MAMMAL
            { eats(animal) ← }
            LION isa CARNIVORE
          HERBIVORE  isa MAMMAL
            { eats(plant)  ← }
            GIRAFFE instanceof HERBIVORE
              { eats(leaf) ← }
            COW instanceof HERBIVORE
        BIRD isa ANIMAL
          { flies() ← }
          PENGUIN instanceof BIRD
            { ¬flies() }
          EAGLE instanceof BIRD
      PLANT isa LIVING_BEING
        LEAF partof PLANT
        GRASS isa PLANT
```

Here we have a number of concepts related by the built-in relations *Isa*, *PartOf* and *InstanceOf*; we note that these concepts have no (explicit) attributes and are defined through some properties, such as *mortal()*, associated with Living_Being, *eats(animal)*, specified for Carnivore, and others. As we will see later in this paper, each concept has a hidden attribute, that we call *self*, used to assign identity to each instance.

Now we enrich the specification of our ontology by adding the following DISJOINTNESS axioms:

← ISA*(X, animal), ISA*(Y, plant), X=Y
← InstanceOf(X, animal), InstanceOf(Y, plant), X=Y

These are constraint expressing that Animal and Plant are disjoint classes (ISA* is used to denote the transitive closure of *Isa*). Likewise, we can state that Carnivore, Herbivore and Bird are disjoint too.

Besides the above explicit axioms, there are some that are implicit, such as:

o the Isa relation is a PARTIAL ORDER (as any other built-in relation):

. Path(X,Y) ← ISA(X,Y)
. Path(X,Y) ← ISA(X,Z), Path(Z,X)
. ← Path(X,X)

o a concept cannot be both a class and an instance

. ← class(X), instance(X)

o only classes can appear in the Isa relation

. ← ISA(X,Y), Instance(X)
. ← ISA(X,Y), Instance(Y)

Next we provide an informal semantics of the above ontology; to this end we have summarised in the following table a number of queries with the respective expected answers.

The keywords for the semantics of hierarchies are INHERITANCE and DEFAULT REASONING. Indeed, as previously stated, properties defined for a concept are inherited by its sub-concepts, unless explicitly negated (to this end we use TRUE NEGATION). For an instance, the property *eats(herbivore, plant)* is inherited by both *giraffe* and *cow*, so that both *eats(giraffe, plant)* and *eats(cow, plant)* hold. However, we apply the principle that specific pieces of information are more reliable than generic ones. So, if we ask "what does giraffe eat?" the answer will be "leaf" (*eats(giraffe, leaf)* is more specific than *eats(giraffe, plant)*), while the question "what does cow eats?" will be answered by "plant" (indeed, no specific information is available for the concept *cow*). Inherited information can be exploited in several ways. For an instance, if we ask whether giraffe eats plants, the answer will be YES. More interestingly, if we ask if cows eat grass, the answer will be MAYBE. This is because the fact *eats(cow, grass)* is not in contradiction with the general knowledge *eats(cow, plant)*, but we have no evidence about its truth. So, we are in presence of a fact that is neither true nor false – it is undefined (*maybe* fact). Likewise, *eats(cow, leaf)* will have answer MAYBE.

Inheritance of properties can be blocked by using true (classical) negation. In our ontology, the general property *fly(bird)* is overwritten by the negative fact *¬flies(penguin)* (here, ¬ is the classical negation symbol); so, the query *flies(penguin)* will be answered NO, while *flies(eagle)* is true.

3 The People Ontology (derived concepts and relations)

In this example we build a simple ontology that represents people, places they leave in and hobbies they enjoy. We start by defining the concepts we are modelling:

. Place
. person(name:string, age:integer, sex:string, father:person; spouse:person; lives:place)
. hobby(name:string)
. city(name:string, people:integer) ISA {place}, PartOf {country}
. country(name:string, capital:city) ISA {place}

QUERY	**EXPECTED ANSWER**	**COMMENTS**
?- mortal(X)	X = living being	
?- mortal(animal)	YES	
?_eats(herbivore,X)	X = plant	
?_eats(giraffe,X)	X = leaf	A more specific information prevails on a general one
?_eats(cow,X)	X = plant	*Eats(plant)* is inherited by animal and no more specific information holds for *cow*
?_eats(cow,grass)	MAYBE	*Grass Isa Plant* and no more specific information is available; so, it might be true
?_eats(cow,leaf)	MAYBE	*Leaf partOf Plant* and no more specific information is available; so, it might be true
?_eats(X,grass)	X = herbivore-MAYBE	
?_eats(giraffe,plant)	YES	
?_eats(carnivore,bird)	MAYBE	
?_flies(X)	X = bird	
?_flies(penguin)	NO	True negation is used to override general information
?_flies(eagle)	TRUE	

Fig. 1. Intuitive semantics of inheritance

Here, Place is a concept with no (explicit) attribute. We assume that it is a concept for which instances are not allowed; thus, to express this condition, we add the following axiom

← Place(self:X)

stating that, for each X, *Place(self:X)* must be false.

The concept *Person* has several attributes, some of which of type concept, namely, *father* and spouse of type *Person*, and *lives* of type *Place*. To state that the marriage relation is symmetric, we add the following axiom

person(self:X, spouse:Y), person(self:Y, spouse:Z), X ¡¿ Z

whose meaning is the following: it cannot happen that a person X has spouse Y and Y has spouse Z and X and Z are different individuals.

Hobby is another concept with a unique attribute. Finally, we have *Country* which is a *Place* and *City* which is a *Place* and is part of a *Country*.

A possible instance of the concept *person* is the following:

Person(p1, Mario, 34, p3, p5, c2)

where *p1* is the object identifier of the instance (it is the value of the attribute *self*), *p3* and *p5* are the object identifiers of the father and the spouse of *p1*, respectively, and *c2* is the object identifier of the city where *p1* lives (note that the values of the attribute *lives*, of type *place*, are either cities or countries).

We now define two non-taxonomic relations, namely, *lives* and *enjoys*:

. lives(psn: person, plc: place)
. enjoys(psn: person, hby: hobby)

So we can say that a person lives in a place (city or country) and that people enjoy hobbies. The following axiom is used to state the CARDINALITY constraint according to which each person lives in *exactly* one place:

lives(person:X, place:Y), lives(person:X, place:Z), Z ¡¿ Y

So far we have used axioms to state constraints; the following example shows how axioms can be used to define DERIVED classes, i.e., classes whose instances are not explicitly specified, but are inferred from their definitions. Next we show some examples:

```
Man() ISA person
{
    Man (self: X) ← person(self: X, sex:Y), Y=”male”
}

Woman() ISA person
{
    Woman (self: X) ← person(self: X, sex:Y), Y=”female”
}
```

Here, each class is defined by a logic rule that provides a formal definition of its meaning and is used to automatically infer its instances. To complete the specification we need to express that Person is generalization of Man and Woman that is both TOTAL (there is no person who is neither a man nor a woman) and DISJOINT (a person is not both a man and a woman):

```
← Person(self:X), not Man(self:X), not Woman(self:X)
← Man(X), Female(X)
```

As another example of derived class, consider the following definition of “cheerful person” as a person who enjoys at least three hobbies:

```
Cheerful-Person (numOfHobbies: integer) ISA {person}
{
    Busy-Person (self: X, numOfHobbies: Z) ←
                person(self: X), %count(Y:enjoys(X,Y)} = Z, Z¿=3
}
```

Note that the body of the above rule contains a special predicate (AGGREGATE predicate) used to count the number of hobbies enjoyed by person X.

ONTO-DLP allows us to specify derived relations as well; for an instance, we define father_in_law as follows:

```
father_in_law (son:person, father:person)
{
    father_in_law(X,Y) ← person(self:X, spouse: person(father:Y))
}
```

Here the definition consists of a unique rule stating that if Y is the father of the spouse of X then Y is the father in law of X. We point out as the structure *person(father:Y)* (called "class term" in Complex-Datalog) allows us to declaratively navigate the instances of person.

It is immediate to see how we can define (derived) classes as union, difference, intersection of other classes; to see an example, we first add to the People Ontology the classes

o Student(registration#:string, faculty:string) ISA Person
o Worker(salary:integer, company:string) ISA Person

and then we define the derived class Student-Worker as the INTERSECTION of the above two:

```
Student-Worker ISA  Student, Worker
{
    Student-Worker(self:X) ← Student(self:X), Worker(self:X)
}
```

Notice that, although no specific attribute is explicitly stated, Student-Worker inherits all the attributes of both Student and Worker (that, in turn, inherit those of Person).

Finally, we provide the definition of a new derived class by using DISJUNCTION; this class is defined as the subset of persons having the following property: it is the minimal set S of persons such that for each two persons who are married at least one in S:

```
MinMarried() ISA person
{
    MinMarried(X) ∨ not_MinMarried(X) ← person(X)
    ← person(self:X, spouse:Y), not MinMarried(X), not MinMarried(Y)
    ⟸ MinMarried(X)
}
```

Here we have a DISJUNCTIVE logic program defining the concept MinMarried; the disjunctive rule

```
MinMarried(X) ∨ not_MinMarried(X) ← person(X)
```

partitions the set of persons into two subsets, MinMarried and not_MinMarried, i.e., it guesses a possible solution; the strong constraint

```
← person(self:X, spouse:Y), not MinMarried(X), not MinMarried(Y)
```

checks the guess, that is, verifies the statement "X and Y are married and none of them belong to MinMarried" to be false; finally, the WEAK CONSTRAINT

$\Longleftarrow$ MinMarried(X)

minimized the number of persons that belong to MinMarried. It holds that each stable model of this program is a possible set of instances of MinMarried.

4 The Financial Consultants (multiple inheritance)

This is an example of multiple inheritance where contradicting pieces of knowledge are inherited; this is a typical case of non-monotonic reasoning, where adding new knowledge may invalidate the previous one.

The example models a financial consultancy where two experts, Expert1 and Expert2, are asked to advise an investor. Briefly:

1. Expert1 suggests to buy shares when the bull rules over the stock market. More precisely, he suggests to buy "dynamic" stocks, using the money gained selling "defensive" ones. In case of bear ruling, he advises to buy Treasury bonds and defensive titles.
2. Expert2 suggests, when the bear rules, not to buy any kind of shares, but taking only Treasury bonds.
3. Both Expert1 and Expert2 think that, if the potential investor is not available to risk, it is preferable not to buy shares.
4. In the end, the investor: if the budget is below a certain threshold, say c1, he does not buy anything; he can risk only if
   - **o** The budget is above the threshold c1
   - **o** He is already covered enough by Treasury bonds.

The assumption is that Expert1 and Expert2 are equally trustable; however, we consider the investor's opinion to be decisive for the final decision.

We note that there exists a conflict among different opinions: when the bear rules, Expert1 suggests defensive shares, while Expert2 advises against buying any kind of shares.

Next we show the ONTO-DLP encoding. For the sake of simplicity, we split stocks into two categories, dynamic and defensive:

```
Stock(dynamic)
Stock(defensive)

EXPERT1
{
   Buy(X) ← bull, stock(X), X = dynamic.
   Sell(X) ← bull, stock(X), X = defensive.
   Buy(bond) ← bear.
```

```
    Buy(X) ← bear, stock(X), X = defensive.
    ⇐= not open_to_risk, titolo(X), buy(X).
}

EXPERT2
{
    buy(X) ← bear, stock(X).
    buy(bond) ← bear.
    ⇐= not open_to_risk, titolo(X), buy(X).
}

INVESTOR
{
    buy(X) ← stock(X), budget(Y), Y ¡ c1.
    open_to_risk ← budget(X), X ¿= c1, investment_bond(Y), Y ¿ c2.
}
```

These concepts are partially ordered by *MoreReliableThan* (¡) as follows: Investor ¡ Expert1, Investor ¡ Expert2. Note that we may think of experts and investors either as instances of the respective classes or as classes themselves.

The rules associated with each concept above state the respective knowledge about the application domain. In particular, the weak constraint

⇐= not open_to_risk, titolo(X), buy(X).

which is common to both experts, encodes point 3 above according to which it is *preferable* not to buy shares if the investor is not open to risk; that is, it states a condition that should *preferably* be satisfied, but *not necessarily.*

SOLUTIONS: let's ponder about the following cases:

1. Hypothesis: **budget ¡ c1, bull**: the budget is below the threshold, so the investor idea (do not buy anything) wins against all the rest; so, ONTO-DLP offers a single solutions, in which the following facts are true: *not open_to_risk, not buy(defensive), not buy(dynamic), sell(defensive).*
2. Hypothesis: **budget ¿= c1, investment in bond ¡= c2, bull**: the investor has enough money to buy; Expert1 suggests, since the bull rules, to buy dynamic stocks and sell defensive ones; Expert2 has no opinion. So, again, a single solution: {*not open_to_risk, buy(dynamic), sell(defensive)*}.
3. Hypothesis: **budget ¿= c1, investment in bond ¡= c2, bear**: the investor has enough money to buy, but cannot risk; since bear rules, both Expert1 and Expert2 suggest to buy bonds. In addition, Expert1 suggests defensive stocks while Expert2 suggests not to buy any. Anyway, since the investor is not open to risk, both experts think it is preferable not to buy stocks (weak constraint); so the Expert2's opinion (do not buy) overcomes Expert1's one; here, we point out how the weak constraint supports one

of two potential contradicting solutions. There is, again, another single solution, in which the true facts are: {*not open_to_risk, buy(bond), not buy (defensive), not buy(dynamic)*}.

4. Hypothesis: **budget ¿= c1, investment in bond ¿ c2, bear**: the investor can buy, and he's open to risk. Both Expert1 and Expert2 suggest bonds. Expert1 suggests also defensive stocks, while Expert2 suggests not to buy stocks at all. There is a clash, not solvable (they are trustable, both), so that the only true fact is *buy(bond)* (note that *buy(defensive)* is undefined).

5 Synonyms

An ontology can be regarded as a "controlled vocabulary" where a concept or a relation may have several different names, one of which is the preferred one. Thus, we can define the following equivalence relation

Synonyms (concept_name: concept, synonym: concept)

associating to each concept its synonyms within the ontology. The following are possible instances:

Synonyms (place, spot)
Synonyms (slot, site).

6 Conclusions

In this paper we have given an informal presentation of a logic-based formalism for ontology specification. This formalism, called ONTO-DLP (where DLP stands for Disjunctive Logic Programming), supports abstraction mechanisms for the specification of concepts, relations and axioms. Concepts have a schema whose attributes may be of type concept. Relations are either taxonomic (*Isa, InstanceOf, PartOf* and *MoreReliableThan* that are built-in) or non-taxonomic (user-defined). Axioms are logic rules that are used to specify the semantics of concepts and relations (properties of concepts, cardinality constraints and algebraic properties of relations, etc.).

ONTO-DLP supports very complex reasoning tasks based on the following basic features:

o Disjunction in the head of rules: as shown in [7, 12], disjunction provides the language with a very high expressive power.
o Constraints of two types: strong and weak [1, 5]; the former are used to state conditions that must be satisfied, while the latter supports the specification of desiderata
o Classical negation, used to explicitly state negative information; as shown in [7], classical negation can be used within hierarchies to override inherited information

o Default reasoning, i.e., the (multiple) inheritance of properties from concepts to sub-concepts based on mechanisms whereby more specific (reliable) information prevails on general one.

Such features make ONTO-DLV a powerful language for ontology specification.

The semantics of ONTO-DLP is based on stable models (to handle disjunction) where facts can be either true, false or undefined (*maybe*); as we have seen, a three-value logic is very intuitive for the purpose of modeling inheritance-based reasoning.

ONTO-DLP is currently being implemented as an extension of the disjunctive logic programming system DLV [5] to take into account extra-logic constructs (hierarchies, inheritance, etc.)

Acknowledgments

This work was supported by the European Commission under project INFOMIX, project no. IST-2001-33570, and under project ICONS, project no. IST-2001-32429.

References

1. Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Strong and Weak Constraints in Disjunctive Datalog. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, number 1265 in Lecture Notes in AI (LNAI), pages 2–17, Dagstuhl, Germany, July 1997. Springer Verlag.
2. Paweł Cholewiński, V. Wiktor Marek, Artur Mikitiuk, and Mirosław Truszczyński. Computing with Default Logic. *Artificial Intelligence*, 112(2–3):105–147, 1999.
3. Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
4. Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. A Deductive System for Nonmonotonic Reasoning. In Jürgen Dix and Ulrich Furbach and Anil Nerode, editor, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, number 1265 in Lecture Notes in AI (LNAI), pages 363–374, Berlin, 1997. Springer.
5. Wolfgang Faber and Gerald Pfeifer. `DLV` homepage, since 1996. `http://www.dlvsystem.com/`.
6. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.
7. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

8. Georg Gottlob. Complexity and Expressive Power of Disjunctive Logic Programming. In M. Bruynooghe, editor, *Proceedings of the International Logic Programming Symposium (ILPS '94)*, pages 23–42, Ithaca NY, 1994. MIT Press.
9. Sergio Greco, Nicola Leone, and Pasquale Rullo. COMPLEX: An Object-Oriented Logic Programming System. *IEEE Transactions on Knowledge and Data Engineering*, 4(4), August 1992.
10. E. Laenens and D. Vermeir. A Fixpoint Semantics for Ordered Logic. *Journal of Logic and Computation*, 1(2):159–185, 1990.
11. N. Leone and P. Rullo. Ordered Logic Programming with Sets. Dec 1993.
12. Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *INFCOMP*, 135(2):69–112, June 1997.
13. Ilkka Niemelä and Patrik Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, volume 1265 of *Lecture Notes in AI (LNAI)*, pages 420–429, Dagstuhl, Germany, July 1997. Springer Verlag.
14. Prasad Rao, Konstantinos F. Sagonas, Terrance Swift, David S. Warren, and Juliana Freire. XSB: A System for Efficiently Computing Well-Founded Semantics. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, number 1265 in Lecture Notes in AI (LNAI), pages 2–17, Dagstuhl, Germany, July 1997. Springer Verlag.