Disjunctive Logic Programs with Inheritance Revisited (A Preliminary Report)

Stefania Costantini[1] Ramón P. Otero[2] Alessandro Provetti[3] Tran C. Son[4]

[1] Università degli Studi di L'Aquila
Dipartimento d'Informatica
Via Vetoio, Loc. Coppito, I-67100 L'Aquila - Italy
stefcost@univaq.it

[2] University of Corunna
AI Lab Dep. da Computacion
E-15071 Corunna, Galicia, Spain
otero@dc.fi.udc.es

[3] Università degli Studi di Messina
Dip. di Fisica
Salita Sperone 31, I-98166 Messina - Italy
ale@unime.it

[4] Department of Computer Science
New Mexico State University
PO Box 30001, MSC CS
Las Cruces, NM 88003, USA
tson@cs.nmsu.edu

Abstract. We argue for a semantical modification of the language DLP$^<$. We show by examples that the current DLP$^<$ representation in some cases does not provide intuitive answers, in particular when applied to inheritance reasoning. We present and discuss an initial modification of DLP$^<$ that yields the expected answers in some examples that we consider significant

1 Introduction

The disjunctive logic program language DLP$^<$ was introduced in [1] for knowledge representation and non-monotonic reasoning. It has been advocated that inheritance reasoning (see e.g. [2, 4]) can be dealt with under the DLP$^<$ framework. Using DLP$^<$, an inheritance network could be represented by a DLP$^<$ program and the answer set semantics of this program specifies the entailment relation of the original network. We demonstrate this by means of an example, written in the DLP$^<$ syntax (precise definitions are in the next section). Consider a taxonomy of animals with their locomotion properties such as $walks$, $swims$, $flies$, or $creeps$. This can be described by the following DLP$^<$ rules:

$$animal\{\ walks(A) \lor swims(A) \lor flies(A) \lor creeps(A) \leftarrow is\_a(A, animal).$$
$$blood\_circulation(A) \leftarrow is\_a(A, animal).\}$$
$$is\_a(pingu, animal).$$

According to the DLP$^<$ semantics, this program has four answer sets [3], in each one *pingu* has exactly one locomotion method.

Let us consider a subclass of *animal*, say *bird*, specified by the following rules:

$bird : animal\{swims(B) \vee \ flies(B) \vee creeps(B) \leftarrow is\_a(B, bird).\}$

$is\_a(pingu, bird).$

Intuitively, the rule describing birds locomotion is more specific than that describing animal locomotion. Thus, the combined theory should have only three answer sets, where *pingu* either swims or flies or creeps, exclusively. On the other hand, in all three answer sets we have $blood\_circulation(pingu)$. The DLP$^<$ semantics also yields this conclusion.

In this paper, we propose several semantically modifications for DLP$^<$ that enhances its usability in inheritance reasoning. In this paper, however, we argue that, for improving the usability of the language, some generalizations should be made, and some unwanted behavior avoided. In particular, we propose some semantic modifications for DLP$^<$ that enhance its usability in inheritance reasoning. The proposed modifications are motivated and illustrated by means of examples. We will begin with a short overview of DLP$^<$. Afterward, we discuss the weakness of DLP$^<$ in knowledge representation, especially in inheritance reasoning, and discuss our initial proposal semantic fix for DLP$^<$.

2 Syntax and Semantics of DLP$^<$

In this section we review the basic definitions of DLP$^<$ [1]. Let us assume a set $\mathcal{V}$ of *variables,* a set Π of *predicates,* a set Λ of *constants,* and a finite partially ordered set of symbols $(\mathcal{O}, <)$, where $\mathcal{O}$ is a set of strings, called *object identifiers,* and $<$ is a strict partial order (i.e., the relation $<$ is irreflexive and transitive).

The definitions of *term*, *atom*, and *literal* are the standard ones, where function symbols are not considered, and $\neg$ is the strong *negation* symbol. A term, atom, literal, rule, or program is *ground* if no variable appears in it. Two literals are *complementary* iff they are of the form p and $\neg p$, for some atom p. Given a literal L, $\neg \cdot L$ denotes[1] the opposite literal. For a set $\mathcal{L}$ of literals, $\neg \cdot \mathcal{L}$ denotes the set $\{\neg \cdot L \,|\, L \in \mathcal{L}\}$.

A rule r is an expression of the form:

$$a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_k, \ not\ b_{k+1}, \ldots, \ not\ b_m$$

where $a_1 \ldots a_n, b_1, \ldots, b_m$ are literals, and not is the *negation as failure* symbol. The disjunction $a_1 \vee \ldots \vee a_n$ is the *head* of *r,* while the conjunction $b_1, \ldots, b_k, \ not\ b_{k+1}, \ldots, \ not\ b_m$ is the *body*. $b_1, \ldots, b_k$, is called the *positive* part of the body of r, and $not\ b_{k+1}, \ldots, \ not\ b_m$ is called the *NAF (negation* as *failure)* part of the body of *r.* We often denote the sets of literals appearing in the head, in the body, in the positive part of the body, and in the *NAF* part of the body of a rule r by *Head(r), Body(r) Body*$^+$*(r),* and *Body*$^-$*(r),* respectively.

Let an *object* o be a pair $\langle oid(o), \Sigma(o)\rangle$ where $oid(o)$ is an object identifier in $\mathcal{O}$ and $\Sigma(o)$ is a (possibly empty) set of rules associated to it.

A *knowledge base* on $\mathcal{O}$ is a set of objects, one for each element of $\mathcal{O}$. Given a knowledge base $\mathcal{K}$ and an object identifier $o \in \mathcal{O}$, the DLP$^<$ *program for o on $\mathcal{K}$* is the set of objects

$$\mathcal{P} = \{(o', \Sigma(o')) \in \mathcal{K} \,|\ o = o' \text{ or } o < o'\}$$

The relation $<$ induces a partial order on $\mathcal{P}$ in the standard way.

[1] Elsewhere the contrary literal is denoted $\overline{L}$.

Informally, a knowledge base can be viewed as a set of *objects* embedding the definition of their properties specified through disjunctive logic rules, organized in a *is_a* (inheritance) hierarchy (induced by $<$). A program $\mathcal{P}$ for an object *o* on a knowledge base $\mathcal{K}$ consists of the subset of $\mathcal{K}$ *reachable from* o in the *is_a*-net.

Thanks to the inheritance mechanism, $\mathcal{P}$ incorporates the knowledge explicitly defined for *o* plus the knowledge inherited from the higher objects. If a knowledge base admits a *bottom* element (i.e., an object less than all the other objects, by the relation **<**), we call it a *program,* since it is equal to the program for the bottom element. In order to represent the membership of a pair of objects (resp., object identifiers) (o_2, o_1) to the transitive reduction of **<** the notation $o_2 : o_1$ is used, to signify that o_2 is a *sub-object* of o_1.

2.1 The semantics of DLP$^<$

Assume that a knowledge base $\mathcal{K}$ is given and an object *o* has been fixed. Let $\mathcal{P}$ be the DLP$^<$ program for *o* on $\mathcal{K}$. The *Universe* $U_\mathcal{P}$ of $\mathcal{P}$ is the set of all constants appearing in the rules. The *Base* $B_\mathcal{P}$ of $\mathcal{P}$ is the set of all possible ground literals that can be constructed from the predicates appearing in the rules of $\mathcal{P}$ and the constants occurring in $U_\mathcal{P}$. Note that, unlike in traditional logic programming the base $B_\mathcal{P}$ of a DLP$^<$ program contains both positive and negative literals. Given a rule *r* occurring in $\mathcal{P}$, a *ground instance* of *r* is a rule obtained from *r* by replacing every variable X in *r* by $\sigma(r)$ where σ is a mapping from the variables occurring in *r* to the constants in $U_\mathcal{P}$. *ground($\mathcal{P}$)* denotes the (finite) multi-set of all instances of the rules occurring in $\mathcal{P}$

A function *obj_of* is defined, from ground instance of rules in *ground($\mathcal{P}$)* onto the set *O* of the object identifiers, associating with a ground instance $\overline{r}$ of *r* the (unique) object of *r.*

A subset of ground literals in $B_\mathcal{P}$ is said to be *consistent* if it does not contain a pair of complementary literals. An *interpretation* $\mathcal{I}$ is a consistent subset of $B_\mathcal{P}$. Under an interpretation $\mathcal{I} \subseteq B_\mathcal{P}$, a ground literal *L is true* if $L \in \mathcal{I}$, *false* otherwise.

Given a rule *r* in *ground($\mathcal{P}$),* the head of *r* is *true* in $\mathcal{I}$ if at least one literal of the head is true w.r.t $\mathcal{I}$. The body of *r* is true in $\mathcal{I}$ if:

(i) every literal in *Body$^+$(r)* is true w.r.t. $\mathcal{I}$, and

(ii) every literal in *Body$^-$(r)* is false v.r.t. $\mathcal{I}$.

Rule *r* is *satisfied* in $\mathcal{I}$ if either the head of *r* is true in $\mathcal{I}$ or the body of *r* is not true in $\mathcal{I}$.

The semantics of overriding. To deal with explicit contradictions, the following definitions – taken from [1] – are needed.

Definition 1. *Given two ground rules r_1 and r_2, we will say that r_1 threatens r_2 on literal L if 1. $L \in Head(r_1)$, 2. $\neg \cdot L \in Head(r_2)$, and 3. obj_of($r_1$) $<$ obj_of(r_2).*

Equivalently, one can say that r_1 and r_2 are conflicting on L (or r_1 and r_2 are in conflict on L).

Definition 2. *Given an interpretation $\mathcal{I}$ and two ground rules r_1 and r_2 such that r_1 threatens r_2 on literal L, we say that r_1 overrides r_2 on L in $\mathcal{I}$ if:*
1. $\neg \cdot L \in \mathcal{I}$ and 2. the body of r_2 is true in $\mathcal{I}$.

A rule r in ground($\mathcal{P}$) is overridden in $\mathcal{I}$ if for each L in Head(r) there exists r_1 in ground($\mathcal{P}$) such that r_1 overrides r on L in $\mathcal{I}$.

The notion of overriding takes care of conflicts arising between conflicting rules. For instance, suppose that both a and $\neg a$ are derivable in $\mathcal{I}$ from rules *r* and *r'*, respectively. If *r* is more specific than *r'* in the inheritance hierarchy, then *r'* is overridden. As a result, *a* should be preferred to $\neg a$ because it is derivable from a rule, *r*, which is more specific and therefore *more descriptive* of the object itself than *r'*.

Definition 3. *Let $\mathcal{I}$ be an interpretation for $\mathcal{P}$. $\mathcal{I}$ is a model for $\mathcal{P}$ if every rule in ground($\mathcal{P}$) is either satisfied or overridden in $\mathcal{I}$. Moreover, $\mathcal{I}$ is a minimal model for $\mathcal{P}$ if no (proper) subset of $\mathcal{I}$ is a model for $\mathcal{P}$.*

Definition 4. *Given an interpretation $\mathcal{I}$ for $\mathcal{P}$, the reduction of $\mathcal{P}$ w.r.t. $\mathcal{I}$, denoted $G(\mathcal{I}, \mathcal{P})$, is the set of rules obtained from ground($\mathcal{P}$) by removing 1. every rule overridden in $\mathcal{I}$; 2. every rule r such that Body$^{-}(r) \neq \emptyset$; 3. the negative part from the bodies of the remaining rules.*

The reduction of a program is simply a set of ground rules. Given a set S of ground rules, $pos(S)$ denotes the positive disjunctive program (called the *positive version of* S), obtained from S by renaming each negative literal $\neg p(\boldsymbol{X})$ as $p'(\boldsymbol{X})$.

Definition 5. *Let $\mathcal{M}$ be a model for $\mathcal{P}$. We say that $\mathcal{M}$ is a DLP$^{<}$ answer set for $\mathcal{P}$ if it is a minimal model of the positive version $pos(G(\mathcal{M}, \mathcal{P}))$ of $pos(G(\mathcal{M}, \mathcal{P}))$. Clearly, $\mathcal{M}$ is inconsistent if it contains both $p(\boldsymbol{X})$ as $p'(\boldsymbol{X})$.*

3 Knowledge Representation with DLP$^{<}$

In [1] it has been argued that DLP$^{<}$ is a suitable knowledge representation language for default reasoning with exceptions. The usefulness of DLP$^{<}$ in different tasks in knowledge representation and non-monotonic reasoning has been demonstrated by the encoding of classical examples of non-monotonic reasoning. The most interesting feature of DLP$^{<}$, as advocated in [1], is the addition of inheritance into the modeling of knowledge. For example, the famous Bird-Penguin example can be represented in DLP$^{<}$ without the conventional *abnormality predicate* as follows.

Example 1. Consider the following program $\mathcal{P}$ with $O(\mathcal{P})$ consisting of three objects *bird, penguin* and *tweety,* such that *penguin* is a sub-object of *bird* and *tweety* is a sub-object of *penguin:*

$bird\{flies\} \qquad penguin : bird\{\neg flies\} \qquad tweety : penguin\{\}$

The only model of the above DLP$^{<}$ program contains $\neg flies$.

Unlike in traditional logic programming, the DLP$^{<}$ language supports two types of negation, that is *strong negation* and *negation as failure.* Strong negation is useful to express negative pieces of information under the complete information assumption. Hence, a negative fact (by strong negation) is true only if it is explicitly derived from the rules of the program. As a consequence, the head of rules may contain also such negative literals, and rules can be conflicting on some literals. According to the inheritance principles, the ordering relationship between objects can help us to assign different levels of acceptance to the rules, allowing us to avoid the contradiction that would otherwise arise.

3.1 Default inheritance in DLP$^<$

As pointed out in [1], the syntax and semantics of DLP$^<$ allow us to capture forms of non-monotonic reasoning and knowledge representation, including inertia and *is*-nets in a rather straightforward way.

For improving its usability however, we believe that some generalization should be made, and some unwanted behavior avoided. The modifications that we propose to DLP$^<$ are illustrated by means of the following examples.

Consider again the knowledge base that defines animals and their possible ways of motion. For birds, the possible ways of locomotion must be defined, which constitute a subset of general ones. Following Buccafurri et al., [1] we define the following knowledge base:

$$P_1 = \begin{cases} animal & \{l_1 : walk \vee swim \vee run \vee fly \leftarrow \ \} \\ bird : animal & \{ \quad \neg swim \leftarrow \qquad\qquad \neg run \leftarrow \ \} \end{cases}$$

The two DLP$^<$ models of this program are $\{\neg swim, \neg run, walk\}$ and $\{\neg swim, \neg run, fly\}$ which implies that a bird either walks or flies but does not swim and does not run. That is, in order to represent the fact that birds swim or fly, it is necessary to state what birds *do not* do, with respect to the general disjunctive rules. Cases that are left, define implicitly what birds are allowed to do, i.e. walk or fly (or maybe both).

We submit that an improvement is needed here, since:

- in many practical cases it is far more concise to list what the features of the object at hand *are*, rather than what they are not;
- a detailed knowledge of ancestor object definition should not be required;
- unwanted behavior arises if one formalizes the example in the intuitive way, as shown by the first example below, and
- unwanted behavior arises in case of multiple inheritance, as illustrated by the second example below.

To illustrate our point, let us consider the direct, intuitive encoding:

$$P_2 = \begin{cases} animal & \{l_1 : walk \vee swim \vee run \vee fly \leftarrow \ \} \\ bird : animal & \{l_2 : walk \vee fly \leftarrow \qquad\qquad\quad \} \end{cases}$$

the latter formulation may appear conceptually equivalent to the former one, and one would expect the semantics to be the same, which is not the case though. Under the DLP$^<$ semantics, P_2 has two models $\{walk\}$ and $\{fly\}$ which indicate that a bird either walks or flies. Notice that these two models can be obtained from the two models of P_1 by removing the negative literals from them. We believe that given the hierarchical property of objects one would prefer P_2 over P_1 for its intuitiveness and that it conforms to the downward refinement technique one uses in software engineering. After all, we are still able to conclude that a bird walks or fly, which is also the intuitive answer.

What happens if we follow the downward refinement technique in describing penguins? Consider the addition of the following, more specific, definitions:

$$\begin{array}{ll} penguin : bird & \{ \quad \neg fly \leftarrow . \quad \neg walk \leftarrow wounded. \quad \neg walk \leftarrow newborn \ \} \\ pimpi : penguin & \{ \quad newborn \leftarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \} \\ pingu : penguin & \{ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \} \end{array}$$

Consider *pingu*, a penguin, who is neither newborn nor wounded. From $walk \vee fly$ in *bird* and $\neg fly$ in *penguin*, we conclude $walk$, which also satisfies l_1. In this case, we say that rule l_1 is *de facto overridden* by l_2. Thus, for *pingu*, DLP$^<$ concludes that it $walk$ and $\neg fly$, which is what we expected.

The fact that rule l_2 cannot override l_1 (Definition 2) since they are not in conflict, gives rise to some unwanted consequences, which we now discuss.

Consider the penguin *pimpi* who is a newborn. From the rule in *penguin*, we can conclude that *pimpi* does not walk and does not fly, i.e., $\neg walk$ and $\neg fly$. Thus, rule l_2 is overridden by the rules in *penguin*. Rule l_1 will not be overridden because there exists no conflicting rule with l_1 on every literal $L \in head(l_1) \setminus head(l_2)$, which are required to override l_1 (Definition 2). This means that we will have answer sets where $pimpi$ runs or swims. Even though the semantics of DLP$^<$ would entail $\neg walk$ and $\neg fly$ for *pimpi*, the existence of answer sets in which *pimpi* runs or swims seems not reasonable in this situation.

As a result, we believe that in this example rule l_2 should override l_1. In general, *disjunctive rules should override those rules in ancestors of which they are a special case.* Moreover, when describing specializations, new knowledge may be added, which is not present in the ancestor. I.e., rule l_2 could for instance be:

$$walk \vee fly \vee run$$

assuming that run is not included in l_1. Still, we think that l_1 should be overridden.

4 A semantics fix for default inheritance

The counter-intuitive results seen for the newborn penguin example above, can be avoided by slight changes in the semantics of overriding. What is being enforced by the new definition of overriding presented here is the fact that *specificity should never be context-independent*, rather, it should always be evaluated w.r.t. interpretations. Some new definitions are in order now.

Definition 6. *A ground rule r_1 is a specialization of rule r_2 if 1. $obj\_of(r_1) < obj\_of(r_2)$, 2. $Head(r_1) \cap Head(r_2) \neq \emptyset$, and 3. $Body(r_1) \subseteq Body(r_2)$.*

It is easy to see that in P_2, l_2 is a specialization of l_1.

Definition 7. *For an interpretation $\mathcal{I}$, and two conflicting ground rules r_1, r_2 in $ground(\mathcal{P})$ such that $L \in Head(r_2)$ (and $/.L \in Head(r1)$) we say that r_1 overrides r_2 on L in $\mathcal{I}$ if: 1. obj_of(r_1) $<$ obj_of(r_2), 2. $\neg \cdot L \in \mathcal{I}$, and 3. the body of r_2 is true in $\mathcal{I}$.*

The definition below is a stricter version of the original definition of overriding presented earlier on. The second condition is new and disallows the newborn penguin counterexample.

Definition 8. *A rule r in ground($\mathcal{P}$) is overridden in $\mathcal{I}$ if one of the following conditions holds:*

(i) either for each $L \in Head(r)$ there exists r_1 in ground($\mathcal{P}$) such that r_1 overrides r on L in $\mathcal{I}$;
(ii) or, there exists a specialization r' of r and r' is overridden in $\mathcal{I}$.

Going back to *pimpi*'s example, we see that rule l_2 is overridden according to condition (i) but under the new definition, also l_1 is because l_2 is a specialization of l_1 and l_2 is overridden (Condition (ii)). Therefore, the new definition ensures that *overriding a rule in an object implies overriding all its less specific ancestors.* Namely, since *pimpi* does not fly nor walks (which is what birds usually do), it won't any more be supposed to perform any less specific form of locomotion (run, swim, etc.). The general conclusion we draw from this example and discussion is that whenever we have two rules whose relation is similar to that of l_1 and l_2 above, which was called de facto overriding, we should make sure that overriding of l_2 also causes overriding of l_1. Hence, no redundant answer set should be generated.

4.1 Multiple inheritance

The knowledge representation style required by DLP$^<$ as it is now, may yield some unwanted behavior when multiple inheritance and updates are used. This section provides another example of how weak DLP$^<$ is in this task. Consider the knowledge base of objects with their color and shapes with the following rules[2]:

$colored\_object$
$\{color(X, red) \vee color(X, yellow) \vee color(X, green) \vee color(X, blue) \leftarrow object(X)\}$
$shaped\_object$
$\{$
$shape(X, cube) \vee shape(X, sphere) \vee shape(X, cone) \leftarrow object(X)$
$volume(X, V) \leftarrow object(X), shape(X, S), formula(X, S, V).$
$formula(X, cube, V) \leftarrow edge(X, L), V = L \times L \times L$
$formula(X, sphere, V) \leftarrow radius(X, R), V = (4 \times L \times L \times L \times \mathit{\Pi})/3$
$formula(X, cone, V) \leftarrow radius(X, R), height(X, H), V = (H \times R \times R \times \mathit{\Pi})/3$
$\}$
$colored\_cube : colored\_object, shaped\_object$
$\{object(c_1). \quad shape(c_1, cube). \quad edge(c_1, 4).\}$
$green\_object : colored\_object \quad \{color(X, green)\}$
$red\_object : colored\_object \quad \{color(X, red)\}$

At the top of this knowledge base, objects are defined in terms of their color, and the definition of objects in terms of their shape. The shape of an object allows one to compute its volume, by applying the appropriate formula. Then, as a particular case there is a cube, denoted as c_1, defined in terms of its shape. In our view, as discussed above, the specification $shape(c_1, cube)$ should override the general disjunctive definition, while the color is still one of those defined in the parent object. In this case, the object inherits from parent objects both the (disjunctive) specification of the possible colors it might assume, and the way of computing the volume.

Now, let us consider defining objects in terms of their color. The disjunctive specification of color should no longer be applicable, while the various choices about shape, and the corresponding formulas for computing the volume, are inherited. However, in DLP$^<$ as it is now, this example should be defined as follows:

$green\_object{:}colored\_object\{\neg color(X, red)\leftarrow \ \neg color(X, yellow)\leftarrow \ \neg color(X, blue)\leftarrow\}$
$red\_object : colored\_object\{\neg color(X, green)\leftarrow \ \neg color(X, yellow)\leftarrow \ \neg color(X, blue)\leftarrow\}$

[2] For the easy of reading, we use the formulas for computing the volume instead of representing them in LP's notation.

Not only is this definition longer and less readable, but it also yields counter intuitive results when augmented for instance by the following definition:

$$\begin{array}{llll} redgreen\_radius\_object : green\_object, red\_object, shaped\_object & & & \\ \{object(s_1) \leftarrow & shape(s_1, sphere) \leftarrow & radius(s_1, 3) \leftarrow & \\ \ object(p_1) \leftarrow & shape(p_1, cone) \leftarrow & radius(p_1, 2) \leftarrow & height(p_1, 3) \leftarrow\} \end{array}$$

Here, there is an object (called $redgreen\_radius\_object$) specifying instances of spheres (namely, s_1) and cones (namely, p_1) which are either red or green. In our view, the inheritance should lead to create, in this object, the disjunctive rule

$$color(X, red) \vee color(X, green).$$

In fact, inheriting the same attribute by multiple sources, means that the attribute may have multiple values (provided they are not mutually inconsistent).

In DLP$^<$ as it is, $redgreen\_radius\_object$ inherits all definitions from its parent objects, i.e.:

$\{\neg color(X, red) \leftarrow \ \ \neg color(X, yellow) \leftarrow \ \ \neg color(X, blue) \leftarrow \ \ \neg color(X, green) \leftarrow\}$

With respect to their union, the general disjunctive rule is completely overridden, and therefore $redgreen\_radius\_object$ turns out to have *no color.*

In the next section, we propose a semantic fix for this problem. We will show that a knowledge base written in this more general and concise form can be transformed into a DLP$^<$ knowledge base, so as to reuse the semantics and the implementation. The difference is in the easier, more intuitive style for the programmer. Consistency and adequacy of the resulting DLP$^<$ knowledge base are guaranteed by the system.

4.2 Addressing multiple inheritance

In what follows we propose a strengthening of DLP$^<$ that allows us to deal with multiple inheritance. We first define a concept called *sibling rules* as follows.

Definition 9. *Two ground rules r_1, r_2 are siblings if:*

1. *obj_of(r_1) $\not<$ obj_of(r_2) and obj_of(r_2) $\not<$ obj_of(r_1),*
2. *r_1 and r_2 are both the specialization of another rule r, and*
3. *$Body(r_1) = Body(r_2)$.*

Intuitively, two rules are siblings if they describe the properties of two (possibly disjoint) sub-classes of an object.

Definition 10. *Given program $\mathcal{P}$, the corresponding enhanced program $\mathcal{P}'$ is defined as follows. Given objects o, o_1, o_2, $o_i = (oid(o_i), \Sigma(o_i))$ where $o < o_1$ and $o < o_2$ and $o_1 \not< o_2$ and rules $r_1 \in \Sigma(o_1)$, $r_2 \in \Sigma(o_2)$ are siblings, add to $\mathcal{P}$ the rule: $Head(r_1) \vee Head(r_2) \leftarrow Body(r_1)$ (where, by definition, $Body(r_1) = Body(r_2)$)*

In the above example, we would add to *redgreen_radius_object* the rule $color(X, red) \vee color(X, green)$ by merging the sibling rules *color(X,red)* and *color(X,green)* (each one with empty body) as we wanted to do. Notice that an interpretation for $\mathcal{P}$ is also an interpretation for $\mathcal{P}'$, since no new atoms are added. Then, a model for $\mathcal{P}$ is obtained as a model of the enhanced version $\mathcal{P}'$.

Definition 11. *Let $\mathcal{I}$ be an interpretation for $\mathcal{P}'$. $\mathcal{I}$ is a model for $\mathcal{P}$ if every rule in ground($\mathcal{P}'$) is satisfied or overridden in $\mathcal{I}$. $\mathcal{I}$ is a minimal model for $\mathcal{P}$ if no (proper) subset of $\mathcal{I}$ is a model for $\mathcal{P}$.*

Accordingly, we have to consider $\mathcal{P}'$ instead of $\mathcal{P}$ when performing the reduction.

Definition 12. *Given an interpretation $\mathcal{I}$ for $\mathcal{P}$, the reduction of $\mathcal{P}$ w.r.t. $\mathcal{I}$, denoted $G(\mathcal{I}, \mathcal{P})$, is the set of rules obtained from ground($\mathcal{P}'$) by removing 1. every rule overridden in $\mathcal{I}$; 2. every rule r such that Body$^-(r) \neq \emptyset$; 3. the negative part from the bodies of the remaining rules.*

5 Conclusions

In this paper we argued, mainly by examples, that to become a viable knowledge representation language that combines the expressiveness of disjunctive logic programming and the convenience of inheritance, DLP$^<$ needs improvements. We showed that overriding in DLP$^<$ is too weak to accommodate a straightforward encoding of classical examples of non-monotonic reasoning. The same is true for the treatment of multiple inheritance. We proposed the strengthening of DLP$^<$ by modifying the notion of overriding and introducing the concept of specialization. To deal with multiple inheritance, we defined the concept of siblings and enhanced programs. The new semantics provides the correct answers in the discussed examples, but we need more work on the actual range of application of DLP$^<$.

References

1. Buccafurri F., Faber W. and Leone N., 1999. *Disjunctive Logic Programs with Inheritance.* Proc. of ICLP'99, pp. 79–93. Long version submitted for publication.
2. Dung P.M. and Son T.C., 1995. *Nonmonotonic inheritance , argumentation, and logic programming.* In Proc. of the 3th Int'l Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'95), pp. 316–329.
3. Gelfond, M. and Lifschitz, V., 1991. *Classical Negation in Logic Programming and Disjunctive Databases,* New Generation Computing 9, 1991: 365–385.
4. Horty J.F., 1994. Some direct theories of non-monotonic inheritance. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic and Artificial Intelligence and Logic Programming*, pages 111–187. Oxford Uni., 1994.