An Analysis of the Computational Complexity of DeLP through Game Semantics

Preliminary Report

Laura A. Cecchi[*][§] lcecchi@uncoma.edu.ar

Pablo R. Fillottrani[†][‡] fillottrani@inf.unibz.it

Guillermo R. Simari[‡] grs@cs.uns.edu.ar

[*]Depto. de Cs. de la Computación - Fa.E.A.
Universidad Nacional del Comahue
Buenos Aires 1400 - (8300) Neuquén - Argentina

[†]Faculty of Computer Science
Free University of Bozen/Bolzano
Piazza Domenicani, 3 - (39100) Bolzano - Italia

[§]Depto. de Cs. Exactas y Naturales - U.A.R.G.
Universidad Nacional de la Patagonia Austral
L.de la Torre 1070 - (9400) Río Gallegos - Argentina

[‡]Depto. de Cs. e Ing. de la Computación
Universidad Nacional del Sur
Av. Alem 1253 - (8000) Bahía Blanca - Argentina

Key Words: Argumentative Systems, Defeasible Reasoning, Logic Programming, Game-based Semantics, Complexity

Abstract

Defeasible Logic Programming (DeLP) is a suitable tool for knowledge representation and reasoning. Its operational semantics is based on a dialectical analysis where arguments for and against a literal interact in order to determine whether this literal is believed by a reasoning agent. The semantics $\mathcal{GS}$ is a declarative trivalued game-based semantics for DeLP that is sound and complete for DeLP operational semantics.

Complexity theory has become an important tool for comparing different formalism and for helping to improve implementations whenever is possible. For these reasons, it is important to investigate the computational complexity and expressive power of DeLP.

In this paper we present a complexity analysis of DeLP through game-semantics $\mathcal{GS}$. In particular, we have determined that computing rigorous consequences is $\mathcal{P}$-complete and that the decision problem "a set of defeasible rules is an argument for a literal under a *de.l.p.*" is in **P**.

1 Introduction

Defeasible Logic Programming (DeLP) is a general argumentation based tool for knowledge representation and reasoning [GS04][1]. Its operational semantics is based on a dialectical analysis where arguments for and against a literal interact in order to determine whether this literal is believed by a reasoning agent. The semantics $\mathcal{GS}$ is a declarative trivalued game-based semantics for DeLP that links game and model theories [CS04]. Soundness and completeness of $\mathcal{GS}$ with respect to DeLP operational semantics have been proved.

Even thought complexity for nonmonotonic reasoning systems has been studied in depth for several formalisms such us default logic, autoepistemic logic, circumscription, abduction and logic programming [CS93, DEGV01] until now not many complexity results for argumentation systems have been reported. This situation can partly be explained by the fact that, historically, implementations of argumentation systems have been limited to the legal area and with no real time response restriction (see [Pol95, Ver98]). However, in recent years, several applications have been developed and implemented using argumentation systems related, for instance, with multiagents systems and web search [ABCMB04, CM04, BAV04].

Complexity theory has become an important tool for comparing different formalism and for helping to improve implementations whenever is possible. For this reason, it is important to analyze computational complexity and expressive power of DeLP, since the former tells us how difficult it is to answer a query, while the latter gives precise characterization of the concepts that it is possible to define as queries.

Even though Dimopoulus *et al.* [DNT02] presents computational complexity of the argumentation based abstract framework for default reasoning [BDKT97], DeLP does not fit in this framework. Another notable study of computational complexity of defeasible systems has been done in [Mah01]. However, the defeasible theory analyzed differs from DeLP in several points, such as the kind of knowledge: facts and strict, defeasible and defeaters rules and its operational semantics.

In this paper we present a complexity analysis of DeLP through game-semantics $\mathcal{GS}$. In particular, we have determined that computing rigorous consequences is **P**-complete and that the decision problem of whether "a set of defeasible rules is an argument for a literal under a defeasible logic program" is in **P**.

The rest of the paper is structured as follows. In section 2, we briefly present basic concepts of DeLP. Section 3, describe the declarative semantics game-based $\mathcal{GS}$. Next we analyze DeLP through $\mathcal{GS}$ semantics pointing out which decision problems must been studied. In section 5, we present the main results of this paper: the complexity analysis of some problems in DeLP. Finally, we present our conclusions and future research lines.

2 DeLP and Operational Semantics Intuitions

We will start by introducing some of the basic concepts in DeLP. In the language of DeLP a literal L is a atom A or a negated atom $\sim A$, where $\sim$ represents the strong negation in the logic programming sense. The complement of a literal L, denoted as $\overline{L}$, is defined as follows: $\overline{L} = \sim A$, if L is an atom, otherwise if L is a negated atom, $\overline{L} = A$.

Definition 1 *A* strict rule *is an ordered pair, denoted "Head $\leftarrow$ Body", whose first member "Head" is a ground literal, and whose second member "Body" is a finite set of ground literals.*

[1]An on-line interpreter of DeLP can be found in `http://lidia.cs.uns.edu.ar/DeLP`

A strict rule with head L_0 and body $\{L_1, \ldots L_n, n > 0\}$ can also be written as $L_0 \leftarrow L_1, \ldots L_n$. If body is the empty set, then we can write L_0. and it is call a Fact. *A* defeasible rule *is an ordered pair, denoted "Head $\prec$ Body", whose first member "Head" is a ground literal, and whose second member "Body" is a finite, non-empty set of ground literals. A defeasible rule with head L_0 and body $\{L_1, \ldots L_n, n > 0\}$ can also be written as $L_0 \prec L_1, \ldots L_n$.*
A defeasible logic program *$\mathcal{P}$, abbreviated de.l.p., is a set of strict rules and defeasible rules. We will distinguish the subset Π of strict rules and the subset Δ of defeasible rules, and we will denote $\mathcal{P} = \langle \Pi, \Delta \rangle$, when required.*

Intuitively, whereas Π is a set of certain and exception-free knowledge, Δ is a set of defeasible knowledge, i.e., tentative information that could be used, whenever nothing is posed against it. Even though a *de.l.p.* may be an infinite set of strict and defeasible rules, complexity analysis has been limited to finite *de.l.p.*.

DeLP operational semantics is based on developments in non monotonic argumentation systems [Pol87, SL92]. An *argument for a literal L* is a minimal subset of Δ that together with Π consistently entails L [GS04]. Thus, an agent can explain a literal L, throughout this argument.

However, in order to determine whether a literal L is supported from a *de.l.p.* a dialectical tree for L is built. An argument for L represents the root of the dialectical tree, and every other node in the tree is a defeater argument against its parent. At each level, for a given a node we must consider all the arguments against that node. Thus every node has a descendant for every defeater. A comparison criteria is needed for determining whether an argument defeats another one. Even though there exists several preference relations considered in the literature, in this first approach we will abstract away from that issue.

We will say that a literal L is warranted if there is an argument for L, and in the dialectical tree each defeater of the root is itself defeated. Recursively, this leads to a marking procedure of the tree that begins by considering that leaves in the dialectical tree are undefeated arguments as a consequence of having no defeaters to consider. Finally, an agent will believe in a literal L, if L is a warranted literal.

There exist four possible answers for a query L: YES if L is warranted, NO if $\overline{L}$ is warranted (i.e., the complement of L is warranted), UNDECIDED if neither L nor $\overline{L}$ are warranted, and UNKNOWN if L is not in the underlying signature of the program.

We have briefly given an intuitive introduction to the DeLP language and the dialectical procedure for obtaining a warranted conclusion. For complete details on DeLP see [GS04]. Now we will continue with the analysis of the declarative semantics $\mathcal{GS}$ for DeLP.

3 Basic Concepts on Game Semantics

Games have an analogy with a dispute and, therefore, that analogy extends to argument-based reasoning. A dispute can be seen as a game where in an alternating manner, the player P, the proponent, starts with an argument for a literal. The player O, the opponent, attacks the previous argument with a counterargument strong enough to defeat it. The dispute could continue with a counterargument of the proponent, and so on. When a player runs out of moves, i.e., that player can not find a counterargument for any of his adversary's arguments, the game is over. If the proponent's argument has not been defeated then she has won the game.

The semantics $\mathcal{GS}$ is a declarative trivalued game-based semantics for DeLP that links game and model theories. Soundness and completeness of $\mathcal{GS}$ with respect to DeLP operational semantics have been proved [CS04]. In the following we present some notions of $\mathcal{GS}$, for more details see [CS00, CS04].

Definition 2 *Let $\mathcal{P}$ be a de.l.p.. A* game-based interpretation for $\mathcal{P}$*, or G-Interpretation for $\mathcal{P}$ for short, is a tuple $\langle T, F\rangle$, such that T and F are subsets of atoms of the underlying signature of $\mathcal{P}$ and $T \cap F = \varnothing$.*

Let *Lit* be the set of all the ground literals that can generated considering the underlying signature of a *de.l.p.*. The set of atoms UNDECIDED is defined as the set $U = Lit^{+} - \{T \cup F\}$, being Lit^{+} the set of all the atoms in *Lit*.

Movements in a game are arguments. For every argument $\mathcal{A}$ for a literal L we can built a game whose first move is $\mathcal{A}$. Thus, a family of games will be obtained considering all the arguments for L. Each game can finish in two possible ways: won by the P or won by the O. There is no place for a draw. As the first move is made by the P, we are interested in those games won by this player.

Definition 3 *Let $\mathcal{P}$ be a plr, L an atom of the underlying signature of $\mathcal{P}$, $\mathcal{F}(L, \mathcal{P})$ the game family for the literal L and $\mathcal{F}(\overline{L}, \mathcal{P})$ the game family for the literal $\overline{L}$ for the plr $\mathcal{P}$. A* game-based model for $\mathcal{P}$*, that we name G-Model of $\mathcal{P}$, is a G-interpretation $\langle T, F\rangle$ such that:*

- *If there exists a game $G(\langle \mathcal{A}, L\rangle, \mathcal{P})$ in the family $\mathcal{F}(L, \mathcal{P})$ won by P, then L belongs to T.*
- *If there exists a game $G(\langle \mathcal{A}, \overline{L}\rangle, \mathcal{P})$ in the family $\mathcal{F}(\overline{L}, \mathcal{P})$ won by P, then L belongs to F.*

Since we only consider literals under the signature of *de.l.p.*, the G-model definition does not contemplate the answer UNKNOWN. The minimal G-model defines a sound and complete semantics $\mathcal{GS}$ for DeLP [CS04].

In order to capture through a game the dialectical procedure, we need to define in a declarative way the movements of such game: the argument. The followings definitions are based on the notation introduced in [Lif96].

Definition 4 [CS00] *Let X be a set of ground literals. The set X is* rigorously closed under a *de.l.p.* $\mathcal{P}$*, if for every strict rule $Head \leftarrow Body$ of $\mathcal{P}$, $Head \in X$ whenever $Body \subseteq X$ and for every defeasible rule $Head' \prec Body'$ of $\mathcal{P}$, $Head' \in X$ whenever $Body' \subseteq X$.*
The set X is consistent *if there is no literal L such that $\{L, \overline{L}\} \subseteq X$. Otherwise, we will say that X is* inconsistent.
We say that X is logically closed *if it is consistent or it is equal to Lit.*

Intuitively, if the set of knowledge of an agent is rigorously closed under a *de.l.p.*, the agent will not believe in a literal that she cannot explain.

Definition 5 [CS00] *Let Ψ a set of strict and defeasible rules. The* set of rigorous consequences *under the rules Ψ, that we will denoted $Cn_R(\Psi)$, is the least set of literals w.r.t. the inclusion, such that it is logically closed and rigorously closed under Ψ.*
Let $\mathcal{P}$ be a de.l.p., the set of rigorous consequences *of $\mathcal{P}$ is defined as $Cn_R(\mathcal{P})$.*

Even though rigorous consequences do not reflect the underlying ideas of strict and defeasible rules, it is very useful for introducing a new definition of argument.

Definition 6 [CS00] *Let $\mathcal{P} = \langle \Pi, \Delta \rangle$ a de.l.p.. We say that $\langle \mathcal{A}, L \rangle$ is an argument structure for a ground literal L, if $\mathcal{A}$ is a set of defeasible rules of Δ, such that:*

1. $L \in Cn_R(\Pi \cup \mathcal{A})$
2. $Cn_R(\Pi \cup \mathcal{A}) \neq Lit$
3. *$\mathcal{A}$ is w.r.t. inclusion, i.e., there is no $\mathcal{A}' \subseteq \mathcal{A}$ such that satisfies (1) and (2).*

We have briefly presented the DeLP language and its operational and declarative game-based semantics. Now, we will be able of analyzing the system and studying its complexity.

4 Discussion on $\mathcal{GS}$ Complexity

There are four ways to measure the complexity of evaluating queries in a specific language [Var82, PY97, DEGV01]:

- **Data complexity:** a specific query in the language is fixed and we study the complexity of applying this query to arbitrary databases. The complexity is then given as a function of the size of the databases.
- **Program complexity or expression complexity:** a specific database is fixed and we study the complexity of applying queries represented by arbitrary expressions in the language. The complexity is given as a function of the length of the expression.
- **Combined complexity:** considers both query and database instance as input variables. Combined complexity is rarely differentiated from expression complexity.
- **Parametric complexity:** complexity is expressed as function of some reasonable parameters. For instance, the number of variables appearing in the query [PY97].

The main complexity measure for logic programming is combined complexity. Hereafter, when we refer to the complexity of a fragment of logic programming, we mean the following type of complexity [DEGV01]:

Complexity of (some fragment of) logic programming: is the complexity of checking if for variable programs P and variable ground atoms A, $P \models A$.

Consequently, as we are interested in computing the complexity of defeasible logic programming, and following the above principle, we will investigate if a query A is warranted by a *de.l.p.* $\mathcal{P} = (\Pi, \Delta)$, denoted $\Pi \models_\Delta A$, adapting [Cap03] notation.

DeLP is an skeptical reasoning system since every consequence of a defeasible logical program is analyzed considering all the arguments for and against it. The semantics $\mathcal{GS}$, a trivalued game semantics, characterizes such skeptical reasoning by the set T and F, since $T \cup \overline{F}$ is the set of all warranted literals, where $\overline{F}$ is the set of the complements of every member of F. Undecided literals are those literals L for which there is no warrant for L or for the complement of L. In this case we must contemplate two possibilities:

- The literal has no argument neither for nor against it, i.e., the agent has no information about the query.

- The literal has an argument for or against it, but it is defeated.

Thus when considering DeLP in relation to game semantics, there are two relevant computational problems to analyze:

- Deciding whether a formula α belongs to the set T or to the set F of a game $G(\langle \mathcal{A}, L\rangle, \mathcal{P})$ won by the proponent P.

- Deciding a formula α belongs to the set $U = Lit^+ - \{T \cup F\}$ of undecided atoms in the G-model.

The former problem, hereafter called GAMESAT, involves finding just a game won by the proponent, but in order to capture the latter it is necessary to find all the games for the literal and for the complement of this literal and to prove that none of them is won by the proponent.

Arguments and counterarguments are the movements in the game. Thus, for analyzing the complexity we study the following decision problem: is a given subset $\mathcal{A}$ of Δ an argument for a literal L? Accordingly with the definition of argument this problem has three parts: is L a consequence of $\Pi \cup \mathcal{A}$?, is $\Pi \cup \mathcal{A}$ consistent?, and is there a subset $\mathcal{A}'$ of $\mathcal{A}$ such that it is consistent with Π and that together with Π derives L? In the next sections we present the main results of this work: this problem is in **P**; furthermore, deciding whether a literal is a consequence of a set of defeasible rules union Π is **P**-complete.

5 The Complexity of Computing an Argument

Building arguments is the computational core of DeLP, for this reason this work analyzes the complexity of the following decision problem: whether a given subset of defeasible rules is an argument for a literal under a *de.l.p.*. As mentioned above, this problem can be divided in three parts according to argument definition.

Let $\mathcal{P} = \langle \Pi, \Delta \rangle$ be a *de.l.p.*, L be a literal and $\mathcal{A} \subseteq \Delta$. First condition of definition 6, that involves rigorous consequences concept is $h \in Cn_R(\Pi \cup \mathcal{A})$. In [CS00], we have defined a transformation Φ from a *de.l.p.* into a definite logic program, i.e., a logic program with just horn clauses. Briefly, Φ transforms negated atoms in new atoms without negation, $\Phi(H \prec B) = \Phi(H) \leftarrow \Phi(B)$ and all other rules or atoms remain the same. We will use this transformation and the following lemma in order to reduce the rigorous consequences of a *de.l.p.* into consequences of a propositional definite logic programming.

Lemma 1 *Let DP be definite logic program and $\mathcal{M}$ be the minimal model of DP, then $\mathcal{M} = Cn_R(DP)$.*

We are interested in computing the time complexity of verifying whether $L \in Cn_R(\Pi \cup \mathcal{A})$. We shall construct a logic program with just Horn clauses, denoted $\mathcal{HP}(\Pi, \mathcal{A}, L)$ such that $h \in Cn_R(\Pi \cup \mathcal{A})$ if and only if $\mathcal{HP}(\Pi, \mathcal{A}, L) \models yes$.

Suppose that $L_1, \ldots, L_n$ are all the literals in $\Pi \cup \mathcal{A}$. We define $\mathcal{HP}(\Pi, \mathcal{A}, L)$ as follows:

$$\mathcal{HP}(\Pi, \mathcal{A}, L) = \Phi(\Pi) \cup \Phi(\mathcal{A}) \cup \{yes \leftarrow \Phi(L)\} \cup \{yes \leftarrow \Phi(L_i), \Phi(\overline{L_i}) : 1 \leq i \leq n\}$$

Even though SAT decision problems, i.e., if there is a truth assignment that satisfies a collection of clauses, is **NP**-complete [GJ79], both checking whether a definite propositional logic program $\mathcal{DP}$ satisfies a ground atom A, i.e., $\mathcal{DP} \models A$, and HORNSAT, i.e., the decision problem of whether or not there is a truth assignment that satisfies a collection of horn clauses, are **P**-complete [DEGV01, PY97].

Lemma 2 *$\mathcal{HP}(\Pi, \mathcal{A}, L)$ is a reduction from $L \in Cn_R(\Pi \cup \mathcal{A})$ into propositional logic programming.*

Proof: In order to prove our claim, we have to establish that:

1. $L \in Cn_R(\Pi \cup \mathcal{A})$ if and only if $\mathcal{HP}(\Pi, \mathcal{A}, L) \models yes$.

   We will consider two cases:

   - $\Pi \cup \mathcal{A}$ is consistent.
     $L \in Cn_R(\Pi \cup \mathcal{A})$ if and only if $\Phi(L) \in \Phi(Cn_R(\Pi \cup \mathcal{A}))$ if and only if $\Phi(L) \in Cn_R(\Phi(\Pi \cup \mathcal{A}))$(see [CS00]) if and only if, by theorem 1, $\Phi(L)$ is in the minimal model of $\Phi(\Pi \cup \mathcal{A})$ if and only if $\mathcal{HP}(\Pi, \mathcal{A}, L) \models yes$ by the definition of minimal model, the monotonicity property and the use of the rule $yes \leftarrow \Phi(L)$.
   - $\Pi \cup \mathcal{A}$ is inconsistent.
     $L \in Cn_R(\Pi \cup \mathcal{A}) = Lit$ if and only if there exists $i, 1 \leq i \leq n$, such that $\Phi(L_i)$ and $\Phi(\overline{L_i})$ are in $\Phi(Cn_R(\Pi \cup \mathcal{A}))$ if and only if $\Phi(L_i)$ and $\Phi(\overline{L_i})$ are in the minimal model of $\mathcal{HP}(\Pi, \mathcal{A}, L)$ if and only if $\mathcal{HP}(\Pi, \mathcal{A}, L) \models yes$ by definition of minimal models, monotonicity property and the use of the rule $yes \leftarrow \Phi(L_i), \Phi(\overline{L_i})$.

2. $\mathcal{HP}$ is computed in logarithmic space: the transformation is quite simple and is feasible in logarithmic space, since rules can be generated independently of each other except those of the form $yes \leftarrow \Phi(L_i), \Phi(\overline{L_i})$ which depends on the literal in the input.

Therefore $\mathcal{HP}(\Pi, \mathcal{A}, L)$ is a reduction from $L \in Cn_R(\Pi \cup \mathcal{A})$ into propositional logic programming. ■

Theorem 1 *Let $\mathcal{P} = (\Pi, \Delta)$ a de.l.p., $\mathcal{A} \subseteq \Delta$ and L a literal. Determining whether $L \in Cn_R(\Pi \cup A)$ is* **P***-complete.*

Proof:

- *Membership:* The least fixpoint T_P^∞ of the operator T_P, given a definite logic program P can be computed in polynomial time [Pap94, PMG98, DEGV01]: the number of iterations is bounded by the number of rules plus one. Each iteration step is feasible in polynomial time. Thus finding the minimal model of a logic program with just Horn clauses is in **P** [DEGV01].

  By lemma 2, $L \in Cn_R(\Pi \cup A)$ has been reduced to propositional logic programming. Therefore, $L \in Cn_R(\Pi \cup A)$ is in **P**.

- *Hardness:* Horn rules are strict rules in a *de.l.p.*, and the minimal model of a definite logic program $\mathcal{P}$ is equal to $Cn_R(\mathcal{P})$. Therefore, by applying reduction by generalization, we have reduced $\mathcal{P} \models L$ to $L \in Cn_R(\Pi \cup A)$. Propositional logic programming is **P**-complete [DEGV01]. Therefore, to find if $L \in Cn_R(\Pi \cup A)$ is **P**-complete.

Algorithm: Minimal
Input: $\mathcal{A}$ an argument for a literal L, and Π a set of strict rules.
Output: true if A is a minimal argument for L, false otherwise

```
minimal=true
Aux= $\mathcal{A}$
While minimal and not $Aux = \varnothing$ do
                    select $H \prec B$ ∈Aux
                    $A' = \mathcal{A} - \{H \prec B\}$
                    if $h \in Cn_R(\Pi \cup A')$ then minimal=false
                                               else Aux= Aux - $\{H \prec B\}$
```

Figure 1: Algorithm for verifying if a set of defeasible rules is minimal with respect to set inclusion for deriving a literal L.

■

Until now we have proved that first condition of argumentation definition is **P**-complete. In the rest of this section we will analyze complexity results over a *de.l.p.* $P = (\Pi, \Delta)$, with m atoms, n strict rules and k defeasible rules.

In Figure 1, we present an algorithm for verifying whether a set of defeasible rules is minimal with respect to set inclusion for entail a literal L. Worst case of the minimality condition is consider assuming that the argument has at most k defeasible rules, i.e., Δ is an argument. Computing minimality condition involves k loops verifying that $L \in Cn_R(\Pi \cup A')$, which is in **P**. Thus, this problem is solvable in polynomial time and therefore, it is in **P**.

Finally, to check if the set of defeasible rules is consistent under a *de.l.p.*, we verify that there is no atom such the atom and its complement are members of $Cn_R(\Pi \cup A)$. In the worst case, when $Cn_R(\Pi \cup A)$ is consistent, this algorithm must control every atom in the signature of the *de.l.p.*. Thus, to check if it is consistent is proportional to the number of atoms and therefore is in **P**.

Both cases can be reduced to propositional logic programming. Because of lack of space we will not present in detail this development but we will explain it briefly. We first transform a *de.l.p.* into a set of horn clauses. Strict rules $Head \leftarrow L_1, L_2, \ldots, L_n$ have been transformed into

$$\mathtt{sr}(\mathtt{Head}, (\mathtt{L_1}, \mathtt{L_2}, \ldots, \mathtt{L_n}))$$

and defeasible rules $Head \prec L_1, L_2, \ldots, L_n$ have been transformed into

$$\mathtt{dr}(\mathtt{Head}, (\mathtt{L_1}, \mathtt{L_2}, \ldots, \mathtt{L_n}))$$

where $Head$ is a literal. Every atom negated is transformed into a new arbitrary atom in the program. Horn clauses to manage consistency and minimality are joined in order to find a model that satisfies its heads. We list such an schema of those clauses in the appendix. Minimality is reduced to checking if `minimal(A,L,A)` is a consequence of the definite logic program, i.e., if A is a minimal set given a potential argument A and a literal L. Consistency is reduced to check if `consistent(A)` is a consequence of the definite logic program. It can be shown that the ground definite logic program defined from such schema is sound and complete.

Theorem 2 *The decision problem "is a given subset of defeasible rules an argument for a literal under a de.l.p.?" is in* **P**.

As a consequence of theorem 2, computing every movement in a game is tractable. However, GAMESAT requires to build every argument that is a response against each argument introduced as a defeater of a previous move. Furthermore, we must also take into account the preference criterion between arguments. Unlike the defeasible system analyzed in [Mah01] in which the preference relation is static, DeLP′s comparison criterion is modular. Therefore, depending on what criterion is chosen, computation could become intractable. One of the criterion used as an example is generalized specificity [SGCnS02], and even though its complexity has not been analyzed, we conjecture that is not tractable.

6 Conclusion and Future Work

We have analyzed DeLP through the $\mathcal{GS}$ semantics, pointing out which decision problems must been studied. We have proved that computing rigorous consequences is **P**-complete and we have proved that the decision problem "whether a given subset of defeasible rules is an argument for a literal under a *de.l.p.*" is in **P** and therefore, it is tractable. However we will be required to compute all the arguments for and against a literal for playing games. Considering the nonmonotonic formalisms complexity results [CS93] we believe that this problem is beyond the tractable class.

As future work, we will analyze the complexity of GAMESAT and the decision problem that deals with undecided literals and we will study the data complexity in order to compare the expressive power of DeLP and of other non monotonic formalisms.

Acknowledgments

This research was partially supported by Secretaría General de Ciencia y Tecnología of the Universidad Nacional del Sur, by the Universidad Nacional del Comahue (Proyecto de Investigación 04/E046) and by Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 No. 13096, PICT 2003 15043, PAV 076).

References

[ABCMB04] Katie Atkinson, Trevor Bench-Capon, and Peter Mc Burney. A dialogue game protocol for multi-agent argument over proposals for action. Technical Report ULCS-04-007, Department of Computer Science, University of Liverpool, Liverpool, U.K., 2004.

[BAV04] N. Bassiliades, G. Antoniou, and I. Vlahavas. A defeasible logic reasoner for the semantic web. In *Proc. of the Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 49–64, 2004.

[BDKT97] A. Bondarenko, P.M. Dung, R Kowalski, and F. Toni. An Abstract, Argumentation-Theoretic Approach to Default Reasoning. *Artificial Intelligence*, 93(1-2):63–101, 1997.

[Cap03] Marcela Capobianco. *Argumentación Rebatible en Entornos Dinámicos.* PhD thesis, Universidad Nacional del Sur, 2003.

[CM04] C. Chesñevar and A. Maguitman. ARGUENET: An Argument-Based Recommender System for Solving Web Search Queries. In *Proc. of the 2nd IEEE Intl. IS-2004 Conference*, pages 282–287, Varna, Bulgaria, June 2004.

[CS93] M. Cadoli and M. Schaerf. A survey of complexity results for nonmonotonic logics. *Journal of Logic Programming*, 17:127–160, 1993.

[CS00] Laura A. Cecchi and Guillermo R. Simari. Sobre la Relación entre la Definición Declarativa y Procedural de Argumento. In *VI CACiC*, Ushuaia, 2000.

[CS04] Laura A. Cecchi and Guillermo R. Simari. Sobre la relación entre la Semántica GS y el Razonamiento Rebatible. In *X CACiC - Universidad Nacional de La Matanza*, San Justo - Pcia. de Buenos Aires, 2004.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374 – 425, September 2001.

[DNT02] Yannis Dimopoulos, Bernhard Nebel, and Francesca Toni. On the Computational Complexity of Assumption-based Argumentation for Default Reasoning. *Artificial Intelligence*, 141(1):57–78, 2002.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[GS04] Alejandro J. García and Guillermo R. Simari. Defeasible Logic Programming: An Argumentative Approach. *Theory and Practice of Logic Programming*, 4(1):95–138, 2004.

[Lif96] Vladimir Lifschitz. Foundations of logic programming. In G. Brewka, editor, *Principles of Knowledge Representation*, pages 1–57. CSLI Publications, 1996.

[Mah01] Michael J. Maher. Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming*, 1(6):691–711, 2001.

[Pap94] Christos Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

[PMG98] D. Poole, A. Mackworth, and R. Goebel. *Computational Intelligence : A logical approach.* Oxford University Press, 1998.

[Pol87] John Pollock. Defeasible Reasoning. *Cognitive Science*, 11:481–518, 1987.

[Pol95] John L. Pollock. *Cognitive Carpentry.* Mass: Bradford/MIT Press, Cambridge, 1995.

[PY97] Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries (extended abstract). In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 12–19, New York, NY, USA, 1997. ACM Press.

[SGCnS02] Frieder Stolzenburg, Alejandro J. García, Carlos I. Chesñevar, and Guillermo R. Simari. Computing Generalized Specificity. *Journal of Applied Non-Classical Logics*, 12:1–27, 2002.

[SL92] Guillermo R. Simari and Ronald P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53:125–157, 1992.

[Var82] Moshe Y. Vardi. The complexity of relational query languages. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC82*, pages 137–146, New York, NY, USA, May 1982. ACM Press.

[Ver98] B. Verheij. Argumed - a template-based argument mediation system for laweyers. In J.C. Hage, T. J. Bench-Capon, A.W. Koers, C.N.J. de Vey Mestdagh, and C.A. Grütters, editors, *Legal Knowledge Based Systems. JURIX: The Eleventh Conference*, pages 113–130, 1998.

Appendix: Horn clauses to manage minimality and consistency

Reduction of minimality problem into propositional logic programming

```
% cons(L,A): is true whenever a literal L is consequences of the argument A
% and the strict rules.
cons(L,A):-sr(L,B),consBody(B,A).
cons(L,A):-member(dr(L,B),A),consBody(B,A).

% consBody(B,A): is true if the body B of a rule is consequence of A
% and the strict rules.
consBody(true,_).
consBody((A,B),RR)<-- cons(A,RR),consBody(B,RR).
consBody(A,RR)<-- A\=(_B,_C),cons(A,RR).

%nocons(L,A) is true if L is not a consequence of an argument A.
nocons(L,A):-nomember(dr(L,B),A),pi(Pi),nomember(sr(L,B),Pi).
nocons(L,A):-member(dr(L,B),A),noconsBody(B,A).
nocons(L,A):-sr(L,B),B\=true,noconsBody(B,A).

% noconsBody(B,A): is true if he body B is not consequence of an argument A
% and the strict rules.
noconsBody((A,_B),RR)<-- nocons(A,RR).
noconsBody((A,B),RR)<-- cons(A,RR),noconsBody(B,RR).
noconsBody(A,RR)<-- A\=(_B,_C),nocons(A,RR).

% del_rule(AP,AP,L,A): is true if  argument A, included in AP, has no
% superfluous rule.
del_rule([],_,_,[]).
del_rule(R,[],_,R).
del_rule([R|RR],RC,L,A)<-- cons(L,RR),delete(R,RC,RD),del_rule(RR,RD,L,A).
del_rule([R|RR],RC,L,A)<-- nocons(L,RR),append(RR,[R],RI),delete(R,RC,RD),
                           del_rule(RI,RD,L,A).

% minimal(AP,L,A): is true if the argument A, included in AP, is a minimal set
% such that the literal L is a consequence.
minimal(AP,L,A):-del_rule(AP,AP,L,A).
```

Reduction of consistent problem into propositional logic programming

```
% Complement of a literal with respect to strong negation.
complement(no(L), L).
complement(L, no(L)).

% Consistency: consistent(A) is true if A is consistent
consistent([]).
consistent(Arg)<-- atomCab(Atomos), consArg(Arg,Atomos,Cons),
                   revCons(Cons,Cons).

% revcons(A,B) is true if for every element E in A, E and its complement
% are member of B.
revCons([],_Cons).
revCons([C|RC],Cons)<-- member(C,Cons),complement(C,NC),
                        nomember(NC,Cons),revCons(RC,Cons).

% consArgAux(Arg,Atomos,AtomsCons)is true if AtomsCons contains every atom
% in Atoms such that it is consequence of Arg union strict rules set.
consArgAux(_Arg,[],[]).
consArgAux(Arg,[no(A)|RA],[no(A)|RC])<-- cons(no(A),Arg), consArgAux(Arg,RA,RC).
consArgAux(Arg,[no(A)|RA],RC)<-- nocons(no(A),Arg),consArgAux(Arg,RA,RC).
consArgAux(Arg,[A|RA],[A|RC])<-- cons(A,Arg), consArgAux(Arg,[no(A)|RA],RC).
consArgAux(Arg,[A|RA],RC)<-- nocons(A,Arg),consArgAux(Arg,[no(A)|RA],RC).

% consArgAux(Arg,Atomos,AtomosConsecuencia) is true if AtomsCons contains every
% atom in Atoms, without repeated elements such that it is consequence of Arg
% union strict rules set.
consArg(Arg,Atomos,Cons)<-- consArgAux(Arg,Atomos,ConsArg),
                            delete_repeated(ConsArg,Cons).

% atomCab(Atomos) is true if Atomos contais every atom in the head of a rule.
atomCab(Atomos)<-- litCab([],L),achieve_atoms(L,Atomos).

% litCab(LA,LN) is true if LN contains all literals in the head of
% defeasible and  strict rules.
litCab(LA,LN)<-- dr(C,_B),nomember(C,LA),litCab([C|LA],LN).
litCab(LA,LN)<-- sr(C,_B),nomember(C,LA),litCab([C|LA],LN).
litCab(L,L).

% achieve_atoms(Lit,Atoms) is true if Atoms contains all the atoms
achieve_atoms([],[]).
achieve_atoms([no(A)|R],[A|RA])<-- achieve_atoms(R,RA).
achieve_atoms([A|R],[A|RA])<-- achieve_atoms(R,RA).
```