Strong Properties of Circumscriptive Logic Programming

Pablo R. Fillottrani* Guillermo R. Simari

Grupo de Investigación en Inteligencia Artificial
Instituto de Ciencias e Ingeniería de Computación
Departamento de Ciencias de la Computación
Universidad Nacional del Sur
Av. Alem 1253
(8000) Bahía Blanca – Argentina

e-mail: ccfillo@criba.edu.ar

Abstract

Dix [7, 5, 6] introduced a method for classifying semantics of normal logic programs. Some of these properties, called *strong properties*, are adaptations of properties from general nonmonotonic theories. We apply this technique to circumscriptive logic programs [8, 9], an extension of traditional logic programming that incorporates circumscriptive policies in the programs. We show this approach preserves *cumulativity*, although it is not *rational* and *supraclassical*. This suggests circumscriptive logic programs have a correct behavior, mantaining properties from normal logic programs.

*Becario de Perfeccionamiento del CONICET.

Strong Properties of Circumscriptive Logic Programming

1 Introduction

Declarative semantics for negation in logic programming has been an area of active research during last years. It started with the extension of pure PROLOG by an operator of *negation as failure*, which admits a negative literal in the absence of a proof for its complement. This definition relies on the procedural semantics provided by SLD resolution, and therefore lacks of a declarative formalization. Several proposals like the closed world assumption [28] or Clark's predicate completion [4] and its three-valued counterparts [10, 16], attempted a solution to the problem, but they were not fully satisfactory [24] due to the fact they did not match exactly the pretended intuitions. After this negation was recognized as one of the first nonmonotonic operators, some conections with nonmonotonic reasoning systems were established [19, 25, 27]. These results motivated the definition of new semantics, like perfect models [19, 2], stable models [12], well-founded models [29, 26], etc., that provide suitable solutions. See [3, 24] for an overview.

Additionally, another negation operator for representing explicit negative information was proposed [13, 14, 30]. This operator, called *strong negation* (also "classical negation" in [13]), handles negative literals in a symmetric way as positive literals: in order to be true both have to be proved. Therefore this semantics is monotonic and can be regarded as if negative literals were syntactic variants of atoms, with the addition of a simple mechanism for consistency preservation.

Neither of these two general types of negation can be ignored. They correspond to rather different intuitions, so problems suitable to be solved with one of them do not have satisfactory solutions with the other. In general, this problem is handled allowing two different syntactic connectives [13, 30]. In [8, 9] *Circumscriptive Logic Programs* were defined to allow the programmer specify the pretended semantics for each negative literal using a single negation connective. These programs incorporate clauses that define a *circumscriptive policy*, specifying whether to apply negation as failure or strong negation to each literal. Thus the meaning of an occurrence of a negative literal depends on this policy, affecting the semantics of the whole program. Lifschitz's pointwise circumscription [18] provides the formal background for these programs.

In view of the different semantics proposed for negation as failure, each one supported by different intuitions, Dix developed in [7, 5, 6] a framework for classifying and characterizing them. This work, following the research of Kraus, Lehmann, Magidor and Makinson in general nonmonotonic theories [15, 21], presents a series of properties and shows whether each of the semantics satisfies them. These properties can be di-

vided into two types: *strong properties* which are adaptations of those introduced for nonmonotonic theories; and *weak properties* which are specifically defined for negation in logic programming. Following this idea, general desired principles for semantics can be identified, providing formal reasons for an irregular behaviour in some of them.

In this paper we study the behaviour of circumscriptive logic programs by applying Dix's framework, in particular regarding strong properties. First we give a brief introduction to the syntax and semantics of these programs, and a statement of the strong properties considered. Then, we present positive results, *i.e.* those properties that are satisfied by the semantics, suggesting it behaves regularly. Next, we show some properties this semantics fails to verify, which do not also hold for well established semantics for negation as failure such as stable and well-founded models. This work will be complemented by a forthcoming paper considering how this semantics behaves regarding the aforementioned weak properties..

2 Circumscriptive Logic Programs

In this section we review the basic concepts of Circumscriptive Logic Programs [8], starting with syntactic considerations.

Definition 2.1 Let $\mathcal{L}$ be a first order language, whose alphabet includes a unary predicate symbol `min` and one n-ary function symbol f_p associated with each n-ary predicate p. With $\text{Lit}_{\mathcal{L}}$ we denote the set of all ground literals in $\mathcal{L}$. A *circumscriptive clause* in $\mathcal{L}$ is a clause of the form:

$$L_0 \leftarrow L_1, \ldots, L_n \tag{1}$$

where all $L_i, 0 \leq i \leq n$ are literals in $\mathcal{L}$. If $n = 0$ the circumscriptive clause is called a *fact*.

A *Circumscriptive Logic Program* P in a language $\mathcal{L}$ is a set of circumscriptive clauses in $\mathcal{L}$ that includes the fact

$$\texttt{min}(f_{\texttt{min}}(X)) \leftarrow \tag{2}$$

Following PROLOG's notation, we will write in this paper predicate and function symbols starting with a lowercase letter and variable symbols with an uppercase letter. The only difference between circumscriptive logic programs and normal logic programs [20] is the possible occurrence of negative literals in the head of the clauses, together with the requirements in the language. Predicate `min` and functions f_p will be used for specifying the *circumscriptive policy* in a logic program, so as to avoid the use of second order logic.

In order to define this circumscriptive policy, we first need to introduce an operation that eliminates certain negative literals in a program. This operation is the relativization of the Gelfond-Lifschitz operator [13] to a given set of ground literals:

Definition 2.2 Let $\mathcal{L}$ be a language. Then if $L \in \text{Lit}_{\mathcal{L}}$ and $S \subseteq \text{Lit}_{\mathcal{L}}$, $L^{\div S}$ is the literal:

$$L^{\div S} = \begin{cases} no\_p(t_1, \ldots, t_n) & \text{if } L = \neg p(t_1, \ldots, t_n) \text{ and } p(t_1, \ldots, t_n) \in S, \\ & \text{being } no\_p \text{ a new } n\text{-ary predicate.} \\ L & \text{otherwise.} \end{cases}$$

This operation can be extended to clauses and programs. If C is a ground circumscriptive program clause of the form (1) then $C^{\div S}$ is the clause $L_0^{\div S} \leftarrow L_1^{\div S}, \ldots, L_n^{\div S}$; and if $P = \{C_i, i \in I\}$ is a circumscriptive logic program then $P^{\div S}$ is the program $P^{\div S} = \{C^{\div S} : C \text{ is a ground instance of a clause } C_i\}$.

We can now define the circumscriptive policy as the definite program resulting from the elimination all negative literals in a circumscriptive logic program.

Definition 2.3 If P is a circumscumscriptive logic program in $\mathcal{L}$, then we define $\text{POL}(P)$, the *circumscriptive policy* of P, as the definite program $P^{\div \text{Lit}_{\mathcal{L}}}$.

If P is a definite program, then M_P denotes its minimal Herbrand model. Thus as $P^{\text{Lit}_{\mathcal{L}}}$ is definite, it has a minimal Herbrand model $M_{P^{\text{Lit}_{\mathcal{L}}}}$. When we mention a circumscriptive logic program P without any reference to a language, we will consider the underlying language formed with all the symbols appearing in P together with those required by definition 2.1. Also, if $t_i, 1 \leq i \leq n$ are terms in the language we will note the term $f_p(t_1, \ldots, t_n)$ as $p(t_1, \ldots, t_n)$ when its usage is clear from the context, in a similar way as meta-programming in logic programs.

The semantics of circumscriptive logic programs is defined from a preference relation between three-valued models, which are called *answer sets*. This preference is originated in a classification of literals according to the circumscriptive policy of a program. First we present this classification of literals and the definition of answer sets, and then we define the preference relation.

Definition 2.4 Let P be a circumscriptive program in a language $\mathcal{L}$. Then if $L \in \text{Lit}_{\mathcal{L}}$, L is called:

- a *default literal* if $L = \neg A$ and $\mathtt{min}(A) \in M_{\text{POL}(P)}$.
- a *variable literal* if $L = A$ or $L = \neg A$ and $\mathtt{min}(A) \notin M_{\text{POL}(P)}$.

Definition 2.5 Let P be a circumscriptive logic program in a language $\mathcal{L}$. Then a set of literals $S \subseteq \text{Lit}_{\mathcal{L}}$ is called an *answer set* of P if and only if it satisfies:

- if $L_0 \leftarrow L_1, \ldots, L_n$ is in P such that $\{L_i, 1 \leq i \leq n\} \subseteq S$, then $L_0 \in S$.
- if there exists in $\mathcal{L}$ an atom A such that $\{A, \neg A\} \subseteq S$ then $S = \text{Lit}_{\mathcal{L}}$.

If S is an answer set of P then $def_P(S) = \{L \in S : L \text{ is a default literal in } P\}$ and $var_P(S) = \{L \in S : L \text{ is a variable literal in } P\}$.

Preference among answer sets of a circumscriptive program will be established from the following pre-order relation.

Definition 2.6 Let P be a circumscriptive logic program in a language $\mathcal{L}$, and $S_1, S_2 \subseteq \text{Lit}_\mathcal{L}$ two answer sets of P. In $P^{\div var_P(\text{Lit}_\mathcal{L})}$ we can consider the dynamic stratification [26] $\{\mathcal{S}_\alpha\}_{\alpha<\delta}$, being δ its depth. We define the relation $S_1 \preceq_P S_2$ if and only if $S_1 \subseteq S_2$ or for each $L \in def_P(S_2) - def_P(S_1)$ there exists $L' \in def(S_1) - def(S_2)$ such that if $L \in \mathcal{S}_\alpha$ and $L' \in \mathcal{S}_\beta$ then $\beta \leq \alpha$.

Definition 2.7 Let P be a circumscriptive logic program in a language $\mathcal{L}$, and $L \in \text{Lit}_\mathcal{L}$. Then we say that L is a *consequence* of P, and write $P \mid\!\approx_{Circ} L$, if and only if it belongs to all $\preceq_P$-minimal answer sets of P. $\mathcal{M}_P$ will denote the set of all consequences of P, *i.e.*, $\mathcal{M}_P = \{L \in \mathcal{L} : P \mid\!\approx_{Circ} L\}$.

The pre-order relation $\preceq_P$ defined over the answer sets of P is similar to the relation $\leq_{P,Z}$ between models of a theory that serves as the basis for the definition of circumscription [22, 17]. It prefers an answer set S_1 over S_2 if S_1 has more default literals than S_2, or if the default literals in S_1 but not in S_2 are preferible to those in S_2 but not in S_1. The consecuence relation $\mid\!\approx_{Circ}$ corresponds to the definition of a priotized circumscriptive theory whose priority is determined by the dynamic stratification of P restricted to default literals.

3 General Properties

This section introduces the abstract properties applied to different semantics for logic programs. These properties are defined [5, 6] for closure operations dependent on each semantics. We first present the general description of this operator.

Definition 3.1 Let P be a logic program in a language $\mathcal{L}$, and $\mid\!\approx_{Sem}$ an entailment relation of a semantics *Sem* defined for all logic programs. We define the *closure operator* for P according to *Sem* as the mapping $C_P : 2^{\text{Lit}_\mathcal{L}} \longmapsto 2^{\text{Lit}_\mathcal{L}}$ where if $U \subseteq \text{Lit}_\mathcal{L}$ then:

$$C_P(U) = \{L \in \text{Lit}_\mathcal{L} : P \cup U \mid\!\approx_{Sem} L\} \tag{3}$$

In (3) $P \cup U$ must be understood as if literals in U were facts, *i.e.*, clauses with empty bodies. In order to keep a logical terminology in the presentation of properties, if $U, V \subseteq \text{Lit}_\mathcal{L}$ we will write $U \wedge V$ for $U \cup V$ and $\neg V$ for the set $\{L : L = \neg A \text{ if } A \in V, \text{ or } L = A \text{ if } \neg A \in V\}$, *i.e.*, the complements of all literals in V.

The main difference with the work of Kraus, Lehman y Magidor [15] on nomonotonic theories is that this operator is only defined for sets of literals, instead of a set of arbitrary formula. This is justified by the fact that program clauses are better understood as inference rules than first order formula. Also, according to this definition the operator $C_P(\cdot)$ accepts infinite sets of literals (like in [21]), whilst the relation in [15] deals only with finite sets.

The technique presented by Dix [7] for classifying and characterizing the various semantics for normal logic programs includes two types of abstract properties. The *strong properties* are adaptations of those for nonmonotonic theories. *Weak properties* reflect the specific behaviour of negation as failure in logic programming. Now we present the main strong properties, applied in this work to circumscriptive logic programs.

Definition 3.2 Let $C_P(\cdot)$ be a closure operation for a semantics Sem for all logic programs P. If $U, V, W \subseteq \mathrm{Lit}_{\mathcal{L}}$ are arbitrary sets of literals then we define the following properties:

Inclusion if $U \subseteq C_P(U)$.

Cut if $V \subseteq C_P(U)$ and $W \subseteq C_P(U \wedge V)$ then $W \subseteq C_P(U)$.

Cautious Monotony if $V \subseteq C_P(U)$ and $W \subseteq C_P(U)$ then $W \subseteq C_P(U \wedge V)$.

Rationality if $\neg V \cap C_P(U) = \emptyset$ and $W \subseteq C_P(U)$ then $W \subseteq C_P(U \wedge V)$.

Distributivity if $C_P(U) \cap C_P(V) \subseteq C_P(U \cap V)$.

Supraclassicality if $Cn(U) \subseteq C_P(U)$, where $Cn()$ denotes the classical consequence operator.

Inclusion is the weaker property, and the one a logic programming semantics should satisfy. *Cut* is one of the most natural conditions desirable for any semantics; it says that by joining consequences to its premises we get no extra inferences. *Cautious monotony* is a kind of converse of *cut*; it expresses that by joining consequences to its premises we lose no inferences. In classical logic this condition is subsumed under monotony. If a consequence operation satisfies both *cut* and *cautious monotony* it is said to be a *cumulative operation*, because if we accumulate conclusions to our premises we do not change inferential power. *Rationality* may be seen in any skeptical semantics (not both of A and $\neg A$ belongs to $C_P(U)$) as a generalization of cautious monotony: we may add any set of formula to our premises without loss of inference as long as it is not contradictory to the consequences. A consequence operation is supraclassical if it includes classical inference; most logic programming semantics are not suprassical because of the fact they are not closed under logical connectives. The condition

of distributivity does not have an inmediate intuitive justification, besides of "its inmense power and its appealing instances" (see Makinson [21]). Note that most of these properties establish conditions that a consequence operator must satisfy after the addition of new data to an existing theory. Therefore, they are also closely related to the logic of Theory Revision [1, 11].

In view of the several approaches to negation as failure, it is important to study how these semantics behave. Sometimes, a new semantics defined to improve an existing one has more serious shortcomings than the original [5]. Most semantics are introduced motivated by an "irregular" behaviour of a previous semantics on a single program [24], being this irregularity justified by the intuitions of the researchers. See [7] for an overview of the properties satisfied by different logic programming semantics.

4 Cumulativity of this Semantics

This section contains our main result, showing circumscriptive logic programs satisfy *cumulativity*. We begin by giving the characterization of the operator $Circ_P : 2^{\mathrm{Lit}_\mathcal{L}} \longmapsto 2^{\mathrm{Lit}_\mathcal{L}}$ as a result of applying definition 3.1 to the inference $\mid\!\approx_{Circ}$ for the semantics defined in section 2.

The following results will be useful in the demonstration of cut and cautious monotony applied to $Circ_P$. The first proposition states that when considering answer sets semantics, it is possible to add or remove facts from the program as long as they belong to the same answer set.

Proposition 4.1 *Let $U, S \subseteq \mathrm{Lit}_\mathcal{L}$ be two sets of literals such that $U \subseteq S$. Then for all circumscriptive logic programs P, S is an answer set of P if and only if S is an answer set of $P \wedge U$.*

Prueba First we note that P and $P \wedge U$ have both the same underlying language, so any answer set of one of them can be an answer set of the other. Suppose now S is an answer set of P. Then, we must prove S satisfies both conditions in definition 2.5 for $P \wedge U$. For the first condition, let $L_0 \leftarrow L_1, \ldots, L_n$ be a circumscriptive clause in $P \wedge U$ such that $\{L_i, 1 \leq i \leq n\} \subseteq S$. If $L_0 \leftarrow L_1, \ldots, L_n \in P$, then $L_0 \in S$ because S is an answer set of P; otherwise if $L_0 \leftarrow \in U$ ($n = 0$ as U only includes facts) then $L_0 \in S$ as $U \subseteq S$. The second condition is inmediate as it does not depend on a particular circumscriptive logic program.

Suppose now S is an answer set of $P \wedge U$. We must prove that S satisfies both conditions in definition 2.5 for P. Again, the second condition is inmediate. Suppose there exists in P a circumscriptive clause $L_0 \leftarrow L_1, \ldots, L_n$ satifying $\{L_i, 1 \leq i \leq n\} \subseteq S$. As $L_0 \leftarrow L_1, \ldots, L_n$ belongs to $P \wedge U$ and S is an answer set of $P \wedge U$ then $L_0 \in S$, so S satisfies the first condition of the definition. Then S is also an answer set of P. ∎

The second result is a theorem proved in [8, chapter 7] saying that the set of consequences of the semantics for circumscriptive logic programs may be considered as an answer set. This is equivalent to saying that the partial order $\preceq_P$ has a minimun answer set for all programs P.

Theorem 4.2 *If P is a circumscriptive logic program in a language $\mathcal{L}$, then the set $\{L \in \mathrm{Lit}_{\mathcal{L}} : P \not\approx_{Circ} L\}$ is an answer set of P.*

We are now able to prove that circumscriptive logic programs satisfy *cut* and *cautious monotony* under the given semantics.

Theorem 4.3 *$Circ_P$ is a cumulative operator for all circumscriptive logic programs P.*

Prueba First we prove $Circ_P$ satisfies *cut*. Suppose $U, V, W \subseteq \mathrm{Lit}_{\mathcal{L}}$ such that $V \subseteq Circ_P(U)$ and $W \subseteq Circ_P(U \wedge V)$. Theorem 4.2 says that $Circ_P(U)$ is an answer set of $P \wedge U$. We can apply the "if" part of proposition 4.1, so $Circ_P(U)$ is an answer set of $P \wedge U \wedge V$. This means that $Circ_P(U \wedge V) \preceq_{P \wedge U \wedge V} Circ_P(U)$. Suppose there exist $L \in Circ_P(U) - Circ_P(U \wedge V)$, then L is a default literal and exists $L' \in Circ_P(U \wedge V) - Cir_P(U)$ such that if $L \in \mathcal{S}_\alpha$ and $L' \in \mathcal{S}_\beta$ then $\beta \leq \alpha$. But then, using the "only if" part of proposition 4.1, $Circ_P(U \wedge V)$ would be an answer set of $P \wedge U$ such that $Circ_P(U \wedge V) \preceq_{P \wedge U} Circ_P(U)$, contradicting the minimality of $Circ_P(U)$. Then $Circ_P(U \wedge V) \subseteq Circ_P(U)$, and $W \subseteq Circ_P(U)$.

We now show that $Circ_P$ satisfies *cautious monotony*. Let $U, V, W \subseteq \mathrm{Lit}_{\mathcal{L}}$ such that $V, W \subseteq Circ_P(U)$. By theorem 4.2 $Circ_P(U \wedge V)$ is an answer set of $P \wedge U \wedge V$. It is obvious that $V \subseteq Circ_P(U \wedge V)$, so applying proposition 4.1 $Circ_P(U \wedge V)$ is an answer set of $P \wedge U$. This means that $Circ_P(U) \preceq_{P \wedge U} Circ_P(U \wedge V)$. Again, if there exist $L \in Circ_P(U \wedge V) - Circ_P(U)$, then L is a default literal and exists $L' \in Circ_P(U) - Cir_P(U \wedge V)$ such that if $L \in \mathcal{S}_\alpha$ and $L' \in \mathcal{S}_\beta$ then $\beta \leq \alpha$. But this would imply, by proposition 4.1, that $Circ_P(U) \preceq_{P \wedge U \wedge V} Circ_P(U \wedge V)$ which is not possible because $Circ_P(U \wedge V)$ is $\preceq_{P \wedge U \wedge V}$-minimum. Then $Circ_P(U) \subseteq Circ_P(U \wedge V)$ and $W \subseteq Circ_P(U \wedge V)$. ∎

Cut is a reasonable property for the purpose of formalizing negation in logic programming. This is also supported by the fact almost all semantics presented satisfies *Cut*, suggesting this condition follow the intuitions behind the idea of negation as failure. *Cautious monotony* is also a nice property. However, there exist some proposals [5] in which it fails. The fact that circumscriptive logic programs have a cumulative consequence operator allow us to affirm it behaves very regularly. The symmetry between these two properties expresses answer sets are stable: our conclusions may be added or removed from the premises with no change in the inference process. Investigation in nonmonotonic formalization shows a natural similarity: Makinson [21] discerns between two clusters of conditions a nonmonotonic logic should satisfy. *Inclusion*, *cut*

and *cautious monotony* precisely form the stronger cluster. These are precisely the properties most logic programming semantics satisfy, including circumscriptive logic programs.

5 Negative Results

In next example we show that *rationality* do not hold in the semantics for circumscriptive logic programs.

Example 5.1 Let P_1 be the following program:

$$\begin{array}{lll} \mathtt{p} & \leftarrow & \neg\mathtt{q} \\ \mathtt{q} & \leftarrow & \mathtt{r} \\ \mathtt{min(q)} & \leftarrow & \end{array}$$

It is clear that $Circ_{P_1}(\emptyset) = \{\mathtt{p}, \neg\mathtt{q}\}$. But $\{\neg\mathtt{r}\} \cap Circ_{P_1}(\emptyset) = \emptyset$ and $Circ_{P_1}(\{\mathtt{r}\}) = \{\mathtt{r}, \mathtt{q}\}$. Then $p \in Circ_{P_1}(\{\mathtt{r}\})$ does not hold as *rationality* would require. This illustrates that the incorporation of a variable literal to our premises may abort the inference of a default literal previously present.

Rationality holds in well-founded semantics and most of its variants, supported model and the three-valued completion semantics. This suggest it is a desirable property. Answer set preferred by $Circ_P$ are much weaker three-valued models due to the presence of variable literals. In fact it is possible to define a restricted form of rationality, in which we are only allowed to add default literals, that holds for $Circ_P$.

Definition 5.2 Let P be a circumscriptive logic program in $\mathcal{L}$, and $U, V, W \subseteq \mathrm{Lit}_{\mathcal{L}}$. Then we say that P satisfies *Restricted Rationality* if $def_P(\neg V) = \neg V$, $\neg V \cap C_P(U) = \emptyset$ and $W \subseteq C_P(U)$ then $W \subseteq C_P(U \wedge V)$.

Thus we define *restricted rationality* as *rationality* applied to a set V containing only default literals. Recall that if $def_P(\neg V) = \neg V$ then $\neg V$ only contains default literals.

Proposition 5.3 *$Circ_P$ satisfies* Restricted Rationality.

Prueba Suppose $U, V \subseteq \mathrm{Lit}_{\mathcal{L}}$ in the conditions given in the definition of the property. If there exist a literal $L \in Circ_P(U)$ such that $L \notin Circ_P(U \wedge V)$, then it is possible to build a new answer set S such that $S \preceq_{P \wedge V} Circ_P(U \wedge V)$ contradicting its minimality. ∎

Another property $Circ_P$ does not satisfy is *supraclassicality*. Again, program P_1 from example 5.1 illustrates this fact: nor p neither ¬p belongs to $Circ_{P_1}(\emptyset)$ so we get $\mathtt{p} \vee \neg\mathtt{p} \notin Circ_{P_1}(\emptyset)$. In the framework of logic programs this property only holds in the

semantics WFS^+ [5]. Most logic programming semantics fail to satisfy it because they consider three-valued semantics or because the semantics is not closed under logical connectives, notably $\leftarrow$.

Distributivity is another property that does not hold under this semantics, as it is shown in the following example:

Example 5.4 Let P_2 be the following circumscriptive logic program:

$$\begin{array}{lll} \mathtt{p} & \leftarrow & \mathtt{a}, \neg\mathtt{b} \\ \mathtt{p} & \leftarrow & \mathtt{b}, \neg\mathtt{a} \\ \mathtt{min(a)} & \leftarrow & \\ \mathtt{min(b)} & \leftarrow & \end{array}$$

Clearly $Circ_{P_2}(\{\mathtt{a}\}) = \{\mathtt{a}, \mathtt{p}, \neg\mathtt{b}\}$ and $Circ_{P_2}(\{\mathtt{b}\}) = \{\mathtt{b}, \mathtt{p}, \neg\mathtt{a}\}$, so literal p belongs to the intersection but it is not in $Circ_{P_2}(\{\mathtt{a}, \mathtt{b}\}) = \{\mathtt{a}, \mathtt{b}\}$.

6 Conclusions

Strong properties for logic programming are conditions over its consequence relation, inherited from the theory of nonmonotonic logic. They are introduced in order to provide a method for characterizing and comparing different logic programming semantics. In this paper we have applied this technique to the semantics of circumscriptive logic programs.

The main result we have proved is that this semantics has a cumulative inference relation, *i.e.* satifies *cut* and *cautious monotony*. Therefore this approach has a regular behaviour in the sense we can modify any set of premises by adding or removing consequences with no change in the conclusions. On the other hand we have shown this semantics fails *rationality*. In general, as several logic programs semantics satisfy this property, this could be considered an inhenrent condition of nonmonotonic negation. As monotony of negation in circumscriptive logic programming depends on its policy, it is not surprinsing *rationality* does not hold in an extension that incorporates both types of negation. Nevertheless, we introduced a new property, called *restricted rationality*, which only considers negation as failure, and we proved it is verified in circumscriptive logic programs.

Among the existing nonmonotonic formalizations, circumscription has played an important role in the definition of logic programming semantics and forms the basis for the characterization of the considered approach. As a nonmonotonic theory circumscription is also a cumulative but not rational nonmonotonic logic. This suggests the intuitions behind circumscription have been succesfully translated to the context of logic programming. This situation is not common in logic programming semantics,

as long as the only cumulative but not rational approach is the O-SEM semantics presented in [23]. Well-founded semantics and most of its versions, all satisfy *cumulativity* and *rationality*.

Other abstract properties this semantics does not verify are *supraclassicality* and *distributivity*. This is not strange in logic programming as a consequence of the fact most semantics are closed under none of the logical connectives ($\wedge$, $\vee$, $\leftarrow$, and not even $\neg$) and then the inference relation has a weaker definition than in a first order theory.

This work will be complemented with the application to this semantics of the weak properties, which treats specifically with negation as failure. The ultimate aim of both types of conditions is to give an abstract characterization of each semantics, and we share this objective for circumscriptive logic programs.

References

[1] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: contraction functions and their associated revision functions. *Journal of Symbolic Logic*, 50, 1985.

[2] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programs*, pages 89–148. Morgan Kaufmann Publishers, Los Angeles, CA, 1988.

[3] K. R. Apt and R. Bol. Logic programming and negation: a survey. *Journal of Logic Programming*, 30, 1993.

[4] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, NY, 1978.

[5] J. Dix. A classification theory of semantics of normal logic programs: I. strong properties. *Fundamenta Informaticae*, 1994.

[6] J. Dix. A classification theory of semantics of normal logic programs: II. weak properties. *Fundamenta Informaticae*, 1994.

[7] J. Dix. Semantics of logic programs: Their intuitions and formal properties. In *Logic, Action and Information*. de Gruyter, 1995.

[8] P. R. Fillottrani. Sistemas de razonamiento no monótono y su relación con la semántica de las bases de datos deductivas. Tesis de Magister en Ciencias de la Computación. Universidad Nacional del Sur, Bahía Blanca, Argentina., 1995.

[9] P. R. Fillottrani and G. R. Simari. Circumscriptive logic programming. In *Proceedings of the XIV International Conference of the Chilean Computer Science Society*, Concepción, Chile, 1994.

[10] M. C. Fitting. A kripke-kleene semantics for general logic programs. *Journal of Logic Programming*, 2, 1985.

[11] P. Gärdenfors. Belief revision and nonmonotonic logic: two sides of the same coin? In *Proceedings of ECAI'90*, Stockholm, Sweden, 1990.

[12] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of ICLP'88*, Seattle, WA, 1988.

[13] M. Gelfond and V. Lifschitz. Classical negation in logic programming and disjunctive databases. *New Generation Computing*, 9, 1991.

[14] R. Kowalski and F. Sadri. Logic programs with exceptions. In *Proceedings of ICLP'90*, Jerusalem, Israel, 1990.

[15] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44, 1990.

[16] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4, 1987.

[17] V. Lifschitz. On the satisfiability of circumscription. *Artificial Intelligence*, 28, 1986.

[18] V. Lifschitz. Pointwise circumscription. In M. L. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 179–193. Morgan Kaufmann Publishers, 1987.

[19] V. Lifschitz. On the declarative semantics of logic programs with negation. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programs*, pages 177–192. Morgan Kaufmann Publishers, Los Angeles, CA, 1988.

[20] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, second, extended edition, 1987.

[21] D. Makinson. General patterns in nonmonotonic reasoning. In D. Gabbay, editor, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume III, pages 35–110. Oxford University Press, 1990.

[22] J. McCarthy. Applications of circumscription to formalizing common-sense reasoning. *Artificial Intelligence*, 28, 1986.

[23] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Contradiction removal within well founded semantics. In *Proceedings of IWLPNR'91*, Washington, DC, 1991.

[24] H. Przymusinska and T. C. Przymusinski. Semantic issues in deductive databases and logic programs. In A. Banerji, editor, *Formal techniques in artificial intelligence*, pages 321–367. North Holland, Amsterdan, 1990.

[25] T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programs*, pages 193–216. Morgan Kaufmann Publishers, Los Angeles, CA, 1988.

[26] T. C. Przymusinski. Every logic program has a natural stratification and an iterated fixed point model. In *Proceedings of the 8th. Symposium on Principles of Database Systems*. ACM SIGACT-SIGMOD, 1989.

[27] T. C. Przymusinski. Three-valued nonmonotonic formalisms and semantics of logic programs. *Artificial Intelligence*, 49, 1991.

[28] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, New York, NY, 1978.

[29] A. van Gelder, K. Ross, and J. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the 8th. Symposium on Principles of Database Systems*. ACM SIGACT-SIGMOD, 1989.

[30] G. Wagner. Logic programming with strong negation and inexact predicates. *Journal of Logic Computation*, 1(6), 1991.