P-stable as an extension of WFS

José Luis Carballido and Claudia Zepeda

Benemérita Universidad Atónoma de Puebla
Facultad de Ciencias de la Computación,
Puebla, Puebla, México
{jlcarballido7,czepedac}@gmail.com

Abstract We summarize the foundations and applications of the new semantics called p-stable and we show that it holds the important property of extending the WFS semantics.

Keywords: WFS semantics, p-stable semantics.

1 Introduction

The research community has long recognized the study of non-monotonic reasoning (NMR) as a promising approach to model features of commonsense reasoning. On the other hand, monotonic logics have been successfully applied as a basic building block in the formalization of non-monotonic reasoning. In this direction, Veloso et al. [27] provide a precise treatment of notions such as 'generally', 'rarely', 'most', 'many', etc. in terms of logics of qualitative reasoning. These (monotonic) generalized logics, with simple sound and complete deductive calculi, are proper conservative extensions of classical first-order logic. Another line of research considers non-classical logics based on some form of logical completions. This paper explores some properties of one of the semantics based on this second formalization.

The stable semantics, which has been successfully used in the modeling of non-monotonic reasoning, was introduced by Gelfond and Lifschitz [11] by means of a simple transformation. More recently, a new semantics useful to model non-monotonic reasoning has been developed: the *p-stable semantics*. The original ideas that motivated this semantics can be found in [16]. Several works on the p-stable semantics have been done, some of the more relevant are [17,18,15,14,4,28,22].

In [18] the authors define the *p-stable semantics* for normal programs in terms of the G'_3 logic and a construct called weak completion.

In [15] the authors generalize to disjunctive programs what was done in [18]. They introduce the p-stable semantics for disjunctive programs by means of a transformation similar to the one used by Gelfond and Lifschitz [11]. Thus, the p-stable semantics for normal and disjunctive programs can be expressed in two ways: in terms of a fixed point operator and classical logic after applying such transformation, and also in terms of the G'_3 logic and weak completions. For

this reason we refer to this semantics with either of the two names: *p-stable* or G'_3*-stable semantics*.

We emphasize that the p-stable semantics presented here, is a two-valued semantics that can be characterized by the three-valued logic G'_3. This semantics should not be confused with the partial stable semantics defined by Przymusinski [24].

This work has two main contributions, the first one corresponds to a summary about the foundations of the new semantics called p-stable which includes its applications and its relationship with paraconsistent logics. The second one corresponds to showing that the p-stable semantics holds the important property of extending the well founded semantics (WFS) defined in [26]. In this way we continue with the study of the p-stable semantics.

The structure of the paper is as follows: Section 2 tells about the foundations of the G'_3-stable semantics and some of its applications. Section 3 starts with basic background and definitions of the G'_3-logic, the p-stable semantics, the X-stable semantics for any logic X and programs with variables. In section 4, we review how the well founded semantics (WFS) can be induced by a powerful method called a *Confluent LP-Systems CS* [9]. Section 5 shows that the p-stable semantics extends the WFS semantics. Then, we present our conclusions.

2 Motivation of the G'_3-stable semantics

This section is dedicated to review the p-stable semantics. We review its origins and foundations, its major known results, and some of its extensions. Although the contribution of this paper is not based on all the results presented in this section, we consider that it is worth to know more about the state of the art of this new semantics.

2.1 Origins and logical foundations of the G'_3-stable semantics

Recently, in [15] an approach for knowledge representation was proposed in terms of any paraconsistent logic stronger than or equal to C_ω, the weakest paraconsistent logic introduced by da Costa [7]. The authors of [14] present a deep study of the paraconsistent logic G'_3 which is a 3-valued logic. The matrices that define the logic G'_3 were originally introduced by Carnielli et al. [6] only to prove that a certain formula is not a theorem in C_w. In fact, the set of theorems of G'_3 is a superset of the set of theorems of C_w.

One interesting feature of the G'_3-logic presented in [14], is that it can be expressed in terms of the Łukasiewicz L_3-logic, and vice-versa, the Łukasiewicz L_3-logic can be expressed in terms of the G'_3-logic. In particular, G'_3 can define the same class of functions as Łukasiewicz 3-valued logic and also can express very directly the strongest intermediate logic (also known as the Gödel's 3-valued logic) G_3. In the same survey, the authors also prove that the logic G'_3 admits a finite axiomatization, in fact, the one presented there consists of the axioms for C_w plus four new axioms.

The authors of [18] introduce the p-stable semantics for normal programs, they also introduce the X-stable semantics for arbitrary programs and for any logic X. The construction used in this definition is called weak completion. The authors present several paraconsistent logics, all of them between C_w, the weakest paraconsistent logic, and Pac, a well known maximal paraconsistent logic studied by Avron [1], and prove that the weak completions of all of these logics are equivalent to each other for normal programs. In [15] the authors establish this last result for disjunctive programs.

Weak completions have been used by D. Pearce [23] to characterize the stable semantics of disjunctive programs. Pearce's result states that weak completions of intermediate constructive logics (in particular G_3 logic and intuitionism) define the stable semantics of disjunctive programs.

In a parallel way, the p-stable semantics of disjunctive programs can be defined in terms of weak completions of paraconsistent logics, in particular G'_3 [17,18]. It can also be expressed in terms of modal logics [17], and it can express the stable semantics of disjunctive programs [15].

We can say then, that two major classes of logics are successfully used to model NMR: constructive intermediate logics and paraconsistent logics. A well known semantics for modeling NMR is the stable semantics, which can be defined in terms of Intermediate logics. This semantics provides a fairly general framework for representing incomplete information and for reasoning with it. The p-stable semantics, which can be defined in terms of paraconsistent logics, shares several properties with the stable semantics, but is closer to classical logic.

2.2 Major Known Results

[15] offers several results about the relation between the stable and the p-stable semantics, in particular every stable model of a normal program is also a p-stable model, but the converse is not true as shown by the program $a \leftarrow \neg a$, which has $\{a\}$ as its unique p-stable model and does not have stable models. The authors also give a sufficient condition on disjunctive programs for a stable model to be a p-stable model, we refer to this condition as "being closed under D-shifts", namely: a disjunctive program P is closed under D-shifts if for any of its disjunctive rules: $\mathcal{H} \leftarrow \mathcal{B}^+, \neg\mathcal{B}^-$, and any $a \in \mathcal{H}$, the rule $a \leftarrow \mathcal{B}^+, \neg\{\mathcal{B} \cup \mathcal{H} - \{a\}\}^-$ also belongs to P. Then we have that any stable model of a disjunctive program closed under D-shifts is also a p-stable model of the program.

One of the main results presented in [15], is the fact that the G'_3-stable semantics is powerful enough to express the well known stable semantics of disjunctive programs; more precisely, the authors present a translation of a disjunctive program D into a normal program N, such that the p-stable model semantics of N corresponds to the stable semantics of D when restricted to the common language of the theories.

The G'_3-stable semantics shares properties with the stable semantics, but is closer to the semantics defined by classical logic. Consider the normal program $P_1: \ \{a \leftarrow \neg b\}$.

The well known stable semantics by Gelfond et al. [11] of P_1, as well as the G'_3-stable semantics give $\{a\}$ as the unique intended model of this program. If we use classical logic we obtain $\{b\}$ as a second model, but this is against the spirit of logic programming.

Now, let us consider the following program P_2: $\{a \leftarrow \neg b, \quad a \leftarrow b, \quad b \leftarrow a\}$.

P_2 does not have stable models, but the set $\{a, b\}$ is a model of P in classical logic. Indeed, this set is also the only G'_3-stable model of P_2.

The main purpose of argumentation theory (Dung [10]), is to study the fundamental mechanism humans use in argumentation, and to explore ways to implement this mechanism on computers. Recently, in [4,19] it was shown that given an argumentation framework, its preferred semantics[1] can be characterized by means of a normal program, such that the preferred extensions of the argumentation framework correspond exactly to the G'_3-stable models of the normal program. The three major semantics of argumentation theory (grounded, stable, and preferred) can be characterized in terms of three logic programming semantics: the well founded semantics (van Gelder et al. [26]), the stable semantics (Gelfond et al. [11]), and the p-stable semantics, respectively, in terms of a unique normal logic program P_{AF}, which is constructed only in terms of the argumentation framework AF. P_{AF} does not depend on any particular semantics. If we want to obtain the stable semantics of AF, we compute the stable semantics of logic programming over P_{AF}. If, on the other hand, we want to obtain the preferred semantics of AF, we compute the p-stable semantics over P_{AF}. Moreover, if we want to obtain the grounded semantics of AF, we compute the well founded semantics over P_{AF}.

These results help to understand the close relationship between two successful approaches to non-monotonic reasoning: argumentation theory and logic programming with negation as failure.

2.3 Expressivity of p-stable semantics

In order to finish this brief motivation, we consider the expressivity of p-stable semantics. We mention three different approaches for knowledge representation based on this semantics: updates, preferences and argumentation (see previous section).

In case intelligent agents get new knowledge and this knowledge must be used to update the knowledge base, it is important to avoid inconsistencies. An update semantics for update sequences of programs based on G'_3-stable semantics is proposed in [20].

The concept of preferences is considered a vital component of reasoning with real-world knowledge. In [21], the authors introduce preference rules which allow to specify preferences as an ordering among the possible solutions to a problem. Their approach allows us to express preferences for arbitrary programs. They

[1] It is worth mentioning that the preferred semantics is one of the most accepted argumentation semantics in argumentation theory. Bench-Capon et al. [2]

also define a semantics for those programs. The formalism used to develop their work is the G'_3-stable semantics.

Finally, we mention that the theory of the p-stable semantics is closely related to topics that have been active areas of research: The theory of paraconsistent logics, the theory of ground non-monotonic modal logics, and the theory of weak completions created by D. Pearce to characterize the stable semantics of disjunctive programs in terms of constructive intermediate logics.

Thus, the G'_3-stable semantics is one of several semantics defined by a family of paraconsistent logics, all of which define the p-stable semantics when restricted to disjunctive programs.

3 Background

Since the present work continues the study of the p-stable semantics, we review the background necessary to understand the definition of the p-stable semantics. In this way, this section summarizes the G'_3 logic, however some different results of the p-stable semantics are related to other logics; we present here some basic background about these logics. We also present some important definitions useful to understand this work.

3.1 Syntax and semantics of logic programs

A *signature* $\mathcal{L}$ is a finite set of elements that we call *atoms*, or *propositional symbols*. The language of a propositional logic has an alphabet consisting of

proposition symbols: $p_0, p_1, \ldots$
connectives: $\wedge$, $\vee$, $\leftarrow$, $\neg$
auxiliary symbols: (,),

where $\wedge$, $\vee$, $\leftarrow$ are 2-place connectives and $\neg$ is a 1-place connective. Formulas are built up as usual in logic. If F is a formula we will refer to its signature $\mathcal{L}_F$ as the set of atoms that occur in F. The formula $F \equiv G$ is an abbreviation for $(F \leftarrow G) \wedge (G \leftarrow F)$. The formula $A \leftarrow B$ is just another way of writing $B \rightarrow A$. A *literal* is either an atom a, or the negation of an atom $\neg a$.

When a formula is constructed as a conjunction (or disjunction) of a set of literals ℓ, $F = \bigwedge \ell$ (or $F = \bigvee \ell$), we denote by $Lit(F)$ such set of literals. A *clause* is a formula of the form $H \leftarrow B$ (also written as $B \rightarrow H$), where H and B, arbitrary formulas in principle, are known as the *head* and *body* of the clause respectively. The body of a clause could be empty, in which case the clause is known as a *fact* and can be noted just by: $H \leftarrow$. In the case when the head of a clause is empty, the clause is called a constraint and is noted by: $\leftarrow B$.

A *normal* clause is a clause of the form $a \leftarrow \bigwedge(\mathcal{B}^+ \cup \neg\mathcal{B}^-)$ where a is an atom, and $\mathcal{B}^+$, $\mathcal{B}^-$ are, possibly empty, sets of atoms. A *disjunctive* clause is a clause of the form $\bigvee \mathcal{H} \leftarrow \bigwedge(\mathcal{B}^+ \cup \neg\mathcal{B}^-)$ where $\mathcal{H}$ is a set of atoms, and $\mathcal{B}^+$, $\mathcal{B}^-$ are, possibly empty, sets of atoms.

The symbol $\neg$, before one of such sets, denotes the conjunction of the negations of the atoms belonging to the set. Sometimes a disjunctive clause may be written as $\mathcal{H} \leftarrow \mathcal{B}^+, \neg\mathcal{B}^-$ following typical conventions for logic programs, similarly for normal clauses. A *definite* program is a normal program whose rules do not have negations in their bodies.

Finally, a *program* is a finite set of clauses. If all the clauses in a program are of a certain type, we say that the program is also of that type. For instance a set of disjunctive clauses is a *disjunctive program*, a set of normal clauses is a *normal program* and so on.

For arbitrary programs, and proper subclasses, we will use $HEAD(P)$ to denote the set of all atoms occurring in the heads of the clauses of P.

Let $Prog_{\mathcal{L}}$ be the set of all normal programs with atoms from the signature $\mathcal{L}$. A *partial interpretation* based on a signature $\mathcal{L}$ is a disjoint pair of sets $\langle I_1, I_2\rangle$ such that $I_1 \cup I_2 \subseteq \mathcal{L}$. A partial interpretation is total if $I_1 \cup I_2 = \mathcal{L}$.

Given two interpretations $I = \langle I_1, I_2\rangle$, $J = \langle J_1, J_2\rangle$, we set $I \leq_k J$ if, by definition $I_i \subseteq J_i$, $i = 1, 2$. Clearly $\leq_k$ is a partial order. We may also see an interpretation $\langle I_1, I_2\rangle$ as a set of literals $I_1 \cup \neg I_2$. When we look at interpretations as sets of literals then $\leq_k$ corresponds to $\subseteq$.

A general semantics SEM is a function on $Prog_{\mathcal{L}}$ which associates with every program a partial interpretation. Given a signature $\mathcal{L}$ and two semantics SEM_1 and SEM_2, we define $SEM_1 \leq_k SEM_2$ if for every program $P \in Prog_{\mathcal{L}}$, $SEM_1(P) \leq_k SEM_2(P)$. When $SEM_1 \leq_k SEM_2$, we say that SEM_2 is an *extension* of SEM_1.

Next, we proceed to give definitions of the relevant logics and semantics.

3.2 The G'_3, C_w and I logics

The G'_3 logic is a 3-valued logic with truth values in the domain $D = \{0, 1, 2\}$ where 2 is the designated value. The evaluation functions of the logic connectives are then defined as follows: $x \wedge y = \min(x, y)$; $x \vee y = \max(x, y)$; the $\neg$ and $\rightarrow$ connectives are defined according to the truth tables given in Table 1.

x	$\neg x$
0	2
1	2
2	0

$\rightarrow$	0	1	2
0	2	2	2
1	0	2	2
2	0	1	2

Table 1. Truth tables of connectives $\neg$ and $\rightarrow$ in G'_3.

We define a tautology as any formula that takes only the designated value 2, regardless of what the truth values the atoms in the formula may take. We will use the notation $\models_{G'_3} A$, to express the fact that A is a tautology in G'_3. We must notice the subtle difference between the logic G'_3 and the best known logic G_3 due to Gödel; the latter is defined exactly by the same functions as the former, except that the negation corresponding to the truth value 1 is 0.

As a consequence, whereas G'_3 accepts the principle of the excluded middle, G_3 does not. An axiomatization for G'_3 is presented in [14]. In particular all of the axioms of C_w are included in such axiomatization. Modus Ponens is the only rule of inference. We will use the notation $\vdash_{G'_3} A$, to express the fact that the formula A is a theorem in G'_3; i.e. A can be inferred from the axioms of G'_3, by using modus ponens.

Positive Logic is defined by the following set of axioms:
Pos 1: $A \rightarrow (B \rightarrow A)$
Pos 2: $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
Pos 3: $A \wedge B \rightarrow A$
Pos 4: $A \wedge B \rightarrow B$
Pos 5: $A \rightarrow (B \rightarrow (A \wedge B))$
Pos 6: $A \rightarrow (A \vee B)$
Pos 7: $B \rightarrow (A \vee B)$
Pos 8: $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow (A \vee B \rightarrow C))$
G'_3 is defined by all axioms of positive logic plus the following six axioms:
1: $A \vee \neg A$
2: $\neg\neg A \rightarrow A$
3: $(\neg A \rightarrow \neg B) \leftrightarrow (\neg\neg B \rightarrow \neg\neg A)$
4: $\neg\neg(A \rightarrow B) \leftrightarrow ((A \rightarrow B) \wedge (\neg\neg A \rightarrow \neg\neg B))$
5: $\neg\neg(A \wedge B) \leftrightarrow (\neg\neg A \wedge \neg\neg B)$
6: $((B \wedge \neg B) \wedge (\sim\sim A \wedge \neg A)) \rightarrow A$
In order to simplify notation, we use $A \vee B$ as an abbreviation for $((A \rightarrow B) \rightarrow B) \wedge ((B \rightarrow A) \rightarrow A)$ in the first of the new six axioms, and in the last axiom we use the abbreviation: $\sim A := A \rightarrow (\neg A \wedge \neg\neg A)$.

One of the main results presented in [14], is a soundness and completeness theorem for the G'_3 logic, namely: a formula is a tautology in the three-valued logic G'_3 if and only if, it is a theorem according to the axiomatization; we can express this in the notation we have introduced: $\models_{G'_3} A$ if and only if $\vdash_{G'_3} A$. We will use any of the two terminologies when referring to such a formula.
Positive logic plus the two axioms: $A \vee \neg A$ and $\neg\neg A \rightarrow A$, define the C_w logic, the weakest paraconsistent logic defined by da Costa [7].

We observe that the formula $(\neg A \wedge A) \rightarrow B$ is not a theorem in either of the two logics. This fact indeed makes these two logics paraconsistent.
The G'_3 logic is strictly stronger than C_w, and both of them are strictly weaker than classical logic. In particular the formula $B \rightarrow [(\neg B \wedge A) \rightarrow C]$, where A,B and C are arbitrary formulas, which is a tautology in classical logic, is not always a tautology in G'_3, as is seen in the particular case when A, B, C take the truth values of 2,1 and 0 respectively. As a useful result we also mention that the formula $(\neg A \rightarrow A) \rightarrow A$ is a tautology in the G'_3 logic for any formula A.
Intuitionistic logic, that we abbreviate as I, is defined as positive logic plus the following two axioms: $(A \rightarrow B) \rightarrow [(A \rightarrow \neg B) \rightarrow \neg A]$ and $\neg A \rightarrow (A \rightarrow B)$. These two axioms allow to do proofs by contradiction but in some limited way; other constructions such as the law of the excluded middle: $(A \vee \neg A)$, are not

valid in intuitionistic logic; however the formula $(\neg A \wedge A) \rightarrow B$ is valid in this logic. All of these properties are shared with the logic G_3 mentioned before; in fact, G_3 is stronger than I in the sense that any theorem in I is also a theorem in G_3.

Notice the opposite situation occurring between intuitionistic and G_3 logics, and the paraconsistent logics C_w and G'_3 regarding the last two formulas above; as it was mentioned before, the first one is valid in C_w and G'_3, but the second one is not. See van Dalen [25] for a good introduction to intuitionistic logic.
We will use the following notation: $\models$ denotes the consequence relation in classical logic, $\vdash_X$ denotes the inference relation in any particular logic X. For any two formulas A and B, $A \equiv_X B$ denotes the fact that A and B are equivalent in logic X, i.e. $A \rightarrow B$ and $B \rightarrow A$ are both theorems or tautologies in logic X, depending on how logic X is defined. Two programs P and Q are equivalent in logic X, denoted: $P \equiv_X Q$, if the conjunction of the rules in P is equivalent, in logic X to the conjunction of the rules in Q.
As a known fact, we observe that the first two axioms of positive logic guarantee the deduction theorem, namely: for Γ, A, B, where Γ is a set of formulas, and A, B are formulas: $\Gamma, A \vdash B$ if and only if $\Gamma \vdash A \rightarrow B$.

3.3 p-stable semantics

From now on we assume that the reader is familiar with the notion of classical minimal model, Lloyd [13].

Here we define the p-stable semantics for disjunctive programs.

Definition 1. *[15] Let P be a disjunctive program and M be a set of atoms. We define: $RED(P, M) = \{H \leftarrow B^+, \neg(B^- \cap M) \mid H \leftarrow B^+, \neg B^- \in P\}$.*

Definition 2. *[15] Let P be a disjunctive program and M be a set of atoms. We say that M is a p-stable model of P if the conjunction of the atoms in M is a logical consequence in classical logic of $RED(P, M)$ (denoted as $RED(P, M) \models M$) and M is a classical model of P (i.e. a model in classical logic).*

Remark 1. If M is a p-stable model of a disjunctive program P, then:
1. $M \subset HEAD(P)$
2. If a fact $a \leftarrow \in P$, then $a \in M$.

1) follows from the fact that $HEAD(P) = HEAD(RED(P, M))$ and the condition $RED(P, M) \models M$; 2) follows from the fact that M is a classical model of P.

For convenience and to be consistent with the definition of a semantics SEM as a function on $Prog_{\mathcal{L}}$ which associates with every program a partial interpretation, from now on, we denote a p-stable model M of a disjunctive program P as a pair $\langle M, \mathcal{L}_P \setminus M \rangle$. We know, that this is not a standard way to represent the p-stable semantics, however this will be useful to present the contribution of this paper.

The following examples illustrate how to obtain the p-stable models of different programs. Our first example shows a disjunctive program with two p-stable models.

Example 1. Let P be the disjunctive program:$\{a \vee b \leftarrow \neg c, \quad a \leftarrow c, \quad b \leftarrow \neg c, \quad c \leftarrow \neg b\}$. Let $M_1 = \{b\}$ and $M_2 = \{a, c\}$. Both sets model (in classical logic) the rules of P. From the definition of the RED transformation we find that $RED(P, M_1) = \{a \vee b \leftarrow, \quad a \leftarrow c, \quad b \leftarrow, \quad c \leftarrow \neg b\}$ and $RED(P, M_2) = \{a \vee b \leftarrow \neg c, \quad a \leftarrow c, \quad b \leftarrow \neg c, \quad c \leftarrow\}$. It is clear that $RED(P, M_1) \models M_1$ and $RED(P, M_2) \models M_2$. Hence $\langle M_1, \{a, c\}\rangle$ and $\langle M_2, \{b\}\rangle$ are p-stable models for P.

The next example shows a program with a single p-stable model, which is also a classical model.

Example 2. Let P be the normal program: $q \leftarrow \neg q$. Let us take $M = \{q\}$ then $RED(P, M)$ is the following program: $q \leftarrow \neg q$. It is clear that M models P in classical logic and $RED(P, M) \models M$ since $(\neg q \rightarrow q) \rightarrow q$ is a theorem in classical logic with the negation $\neg$, now interpreted as classical negation. Therefore M is a p-stable model for P.

Our third example shows a program which has several classical models but has no p-stable models.

Example 3. Let P be the normal program: $\{a \leftarrow \neg b, \quad b \leftarrow \neg c, \quad c \leftarrow \neg a\}$. It is clear that the sets $M_1 = \{a, b\}, \quad M_2 = \{a, c\}, \quad M_3 = \{b, c\}, \quad M_4 = \{a, b, c\}$ are all models of P in classical logic. However, they are not p-stable models of P. If we apply definition 2 to M_4, we have $RED(P, M_4) = P$ and M_4 models P in classical logic, however $RED(P, M_4) \not\models M_4$. If we apply definition 2 to M_1, we obtain $RED(P, M_1)$ as the following program: $\{a \leftarrow \neg b, \quad b \leftarrow, \quad c \leftarrow \neg a\}$ and it is clear that M_1 models in classical logic each of the rules of P. However, the second condition in definition 2 is not satisfied, since $RED(P, M_1) \not\models a$. By symmetry the same result is obtained for M_2 and M_3. Hence, the program does not have p-stable models.

Finally, we present a program which has no stable models and whose p-stable and classical models are the same.

Example 4. Let P be the normal program: $\{a \leftarrow \neg b, \quad a \leftarrow b, \quad b \leftarrow a\}$. We can verify that $M = \{a, b\}$ models the rules of P in classical logic. From the definition of the RED transformation, we find that $RED(P, M) = P$. Now, from the first and third rule, it follows that $(\neg b \rightarrow b)$ where the negation $\neg$ is now interpreted as classical negation. Since $(\neg b \rightarrow b) \rightarrow b$ is a theorem in classical logic, it follows that $P \models M$. Therefore, M is a p-stable model of P.

3.4 The X-stable semantics

Now we review a characterization of the p-stable semantics for disjunctive programs in terms of the G'_3 logic. First we present some useful definitions.

Given a program P and a set of atoms $M \subseteq \mathcal{L}_P$, we call the construct $P \cup \neg M^c$ a *weak completion* of the program P (with respect to the set of atoms M), where the superscript c, denotes set theoretical complement operator with respect to $\mathcal{L}_P$.

Definition 3. *Let P be any theory, X be any logic and M be a set of atoms. M is a X-stable model of P if the next two conditions hold: $\vdash_X P \cup \neg M^c \rightarrow M$ and M is a classical model of P.*

The expression appearing in the first condition of this definition is interpreted as the formula where the antecedent is the conjunction of all rules in P and all literals in $\neg M^c$, and the consequent is the conjunction of all the atoms in M. We will keep using this interpretation in what follows.

Of particular interest to us is the G'_3-stable semantics, which is the result of using the logic G'_3 in the previous definition. For more details see [5].

Example 5. Consider the following logic program: $P = \{b \leftarrow \neg a, a \leftarrow \neg b, p \leftarrow \neg a, p \leftarrow \neg p\}$. It is easy to verify that this program has two G'_3-stable models, which are $\{a, p\}$ and $\{b, p\}$.

Theorem 1 gives a characterization of the p-stable semantics for disjunctive programs in terms of the G'_3 logic. This result was first proven for normal programs in [18]; more recently it has been extended to disjunctive programs in [15].

Theorem 1. *[15] Let P be a disjunctive program and M be a set of atoms. M is a p-stable model of P iff M is a G'_3-stable model of P.*

3.5 Programs with variables

As can be seen, p-stable models are defined for propositional logic programs only. However this definition can be extended to *predicate programs*, which allows the use of predicate symbols in the language, but without function symbols to ensure the ground instance of the program to be finite. So a *term* can only be either a variable or a constant symbol. The ground instance of a predicate program P, $Ground(P)$, is defined in Lifschitz [12] as the program containing all ground instances of clauses in P. Then M is defined as a p-stable model of a predicate program P if it is a p-stable model for $Ground(P)$.

The following section review the WFS semantics in order to show that the p-stable semantics holds the important property of extending the WFS semantics. Since the definition of WFS for normal programs is unique and has been accepted by the research community, as opposed to the case of disjunctive programs, from now on we deal exclusively with normal propositional logic programs.

4 WFS and WFS$^+$ semantics

In this section we review how the well founded semantics (WFS) can be induced by a powerful method called a *Confluent LP-System CS* [9]. This method does

not only characterize well-known semantics for logic programs but can be used to define new semantics. Roughly speaking, a semantics for a class of logic programs determines the set of derivable literals for each program P. The method of determining such a semantics is to rewrite the program P according to certain rewriting rules until we arrive at a normalform of the original program from which we can immediately read off the derivable literals. Suitable sets for rewriting rules correspond to different normalforms and thus to different semantics.

The theory of Confluent LP-Systems CS [9] combines methods from rewriting systems with logic programming technology to define a powerful framework for investigating the semantics of logic programs. The starting point of this theory is to determine a set of rewriting rules that is confluent and that computes the semantics in a canonical way. These rules transform programs into *simpler* programs. *Confluence* and *termination* guarantee that every program has associated with it a *normalform*. This normalform then induces a semantics and a simple and efficient method to answer queries with respect to this semantics.

It is important to note that Confluent LP-Systems CS correspond to a general theory, i.e., the concept is not attached to any particular semantics. [9] shows how most of the classical semantics such as Fitting's 3-valued version of Clark's completion, the wellfounded semantics WFS and Schlipf's extension WFS$^+$ can be defined as a Confluent LP-System in a natural way.

The following definition plays an important role to define the semantics of a normal program based on the notion of a rewriting system [9].

Definition 4. *[9] For any normal program P we define $SEM_{min}(P) = \langle P^{true}, P^{false} \rangle$ where $P^{true} := \{p \mid p \leftarrow \in P\}$, $P^{false} := \{p \mid p \in \mathcal{L}_P \setminus HEAD(P)\}$.*

Now we define a rewriting system, normalform, and some properties of rewriting systems.

Definition 5. *[9] An abstract rewriting system is a pair $\langle S, \rightarrow \rangle$ where $\rightarrow$ is a binary relation on a given set S. Let $\rightarrow^*$ be the reflexive, and transitive closure of $\rightarrow$. When $x \rightarrow^* y$ we say that* x reduces to y*. An irreducible element is said to be in* normalform*. We say that a rewriting system is*
noetherian*: If there is no infinite chain $x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_i \rightarrow x_{i+1} \rightarrow \ldots$, where for all i the elements x_i and x_{i+1} are different,*
confluent*: If whenever $u \rightarrow^* x$ and $u \rightarrow^* y$ then there is a z such that $x \rightarrow^* z$ and $y \rightarrow^* z$.*

In a noetherian and confluent rewritten system, every element x reduces to a unique normalform that we denote by *norm(x)* [9].

The main concept on which the notion of a Confluent LP-Systems CS is based, is the concept of a *transformation* rule.

Definition 6. *[9] A transformation rule is a binary relation on $Prog_{\mathcal{L}}$. Let a program $P \in Prog_{\mathcal{L}}$ be given. We define the following transformation rules.*
RED$^+$*: This transformation can be applied to P, if there is an atom a which does not occurs in $HEAD(P)$.* ***RED***$^+$ *transforms P into the program where all*

occurrences of $\neg a$ are removed.
RED*$^-$: This transformation can be applied to P, if there is a rule $a \leftarrow \in P$. **RED**$^-$ transforms P into the program where all clauses that contain $\neg a$ in their bodies are deleted.*
SUB (Subsumption)*: This transformation can be applied to P, if P contains two clauses $a \leftarrow body_1$, $a \leftarrow body_2$ where $body_1 \subseteq body_2$. **SUB** transforms P into the program where the clause $a \leftarrow body_2$ has been removed.*
Success: Suppose that P includes a fact $a \leftarrow$ and a clause $q \leftarrow body$ such that $a \in body$. Then we replace the clause $q \leftarrow body$ by $q \leftarrow body \setminus \{a\}$.*
Failure: Suppose that P contains a clause $q \leftarrow body$ such that $a \in body$ and $a \notin HEAD(P)$. Then we erase the given clause.*
LC (Logical consequence)*: Suppose $P \models a$ for an atom a. Then we can add the rule $a \leftarrow$ to P.*
Loop: We say that P_2 results from P_1 by Loop w.r.t. A if, by definition, there is a set A of atoms such that*

1. *for each rule $a \leftarrow body \in P_1$, if $a \in A$, then $body \cap A = \emptyset$,*
2. *$P_2 := \{a \leftarrow body \in P_1 \mid body \cap A = \emptyset\}$,*
3. *$P_1 \neq P_2$.*

Although these transformation rules are not really *functions* on $Prog_{\mathcal{L}}$ (e.g. **RED**$^-$ is only determined if an occurrence of a certain rule is distinguished), they induce a set of operators on $Prog_{\mathcal{L}}$ [9]. An operator, denoted as *op*, is a function over the set of programs that transforms a program P into a program P^{op} as follows. If C_1, C_2 are clauses, g is a literal and f is an atom, then *op* can be of type

Red$^+$, then P^{op} is the reduction of P with respect to C_1 and g. We write $\langle \mathbf{RED}^+, C_1, g\rangle$ [9].

We could define in a similar way the operators for the other transformations.

Example 6. [9] If P is $\{c \leftarrow \neg d\}$ and *op*1 is $\langle \mathbf{RED}^+, \neg d \leftarrow,\ c \leftarrow \neg d\rangle$ then P^{op1} is $\{c \leftarrow\}$. If *op*2 is $\langle \mathbf{RED}^+, c \leftarrow \neg d, \neg c\rangle$ then $P^{op2} = P$.

Next, we give the definition of Confluent LP-System CS.

Definition 7. *[9] A* Confluent LP-System CS *over the signature $\mathcal{L}$ is a pair $\langle \{op_i \mid i = 1, \ldots, n\}, \rightarrow\rangle$ that satisfies the following conditions:*
1. $\{op_i \mid i = 1, \ldots, n\}$ is a finite set of transformation rules on $Prog_{\mathcal{L}}$. By abuse of language we often view them as (computable) operators as we explained before.
2. $\langle Prog_{\mathcal{L}}, \rightarrow\rangle$, where $\rightarrow$ is the union of all the transformation rules in CS, is a noetherian and confluent rewriting system.
3. If $P \rightarrow P_1$ then $SEM_{min}(P) \leq_k SEM_{min}(P_1)$.

We denote the uniquely determined normalform of a program P with respect to the Confluent LP-System CS by $norm_{CS}(P)$ [9].

Every Confluent LP-System CS induces a semantics SEM_{CS} as follows [9]: $SEM_{CS}(P) := SEM_{min}(norm_{CS}(P))$.

Now, we present a Confluent LP-System CS$_1$ used to compute the WFS semantics in quadratic time and introduced in [3].

Definition 8. *[3] Let CS_1 be the Confluent LP-System which contains the following transformation rules:* ***RED***$^+$*,* ***RED***$^-$*,* ***Success****,* ***Failure****, and* ***Loop****.*

The Confluent LP-System CS_1 induces the WFS semantics.

Theorem 2. *[9] The WFS semantics is the semantics induced by the Confluent LP-System CS_1.*

The WFS^+ semantics is an extension of WFS introduced in [8]. We present a Confluent LP-System CS_2 used to compute the WFS^+ semantics given in [3].

Definition 9. *[3] Let CS_2 be the Confluent LP-System which contains the following transformation rules:* ***RED***$^+$*,* ***RED***$^-$*,* ***SUB****,* ***TAUT****,* ***LC****,* ***Success****,* ***Failure*** *and* ***Loop****.*

The Confluent LP-System CS_2 induces the WFS^+ semantics.

Theorem 3. *The WFS^+ semantics is the semantics induced by the confluent LP-system CS_2.*

Example 7. Let us obtain the WFS^+ semantics of a normal program. Let us consider the program P: $\{a \leftarrow b,\ a \leftarrow \neg b,\ b \leftarrow a\}$. Applying **LC** we add the rule $a \leftarrow$ to P and we obtain $\{a \leftarrow,\ a \leftarrow b,\ a \leftarrow \neg b,\ b \leftarrow a\}$. Since the last program includes the fact $a \leftarrow$ and the rule $b \leftarrow a$, we can apply **Success** and obtain: $\{a \leftarrow,\ a \leftarrow b,\ a \leftarrow \neg b,\ b \leftarrow\}$. Applying **Success** again we obtain: $P = \{a \leftarrow,\ a \leftarrow \neg b,\ b \leftarrow\}$. Finally, we can delete $a \leftarrow \neg b$ applying $\mathbf{RED}^-$: $P = \{a \leftarrow,\ b \leftarrow\}$. Then the $norm_{CS_2}(P) = \{a \leftarrow,\ b \leftarrow\}$. Thus the WFS^+ semantics of P is $SEM_{CS_2}(P) = SEM_{min}(norm_{CS_2}(P)) = \langle\{a,\ b\}, \emptyset\rangle$.

5 WFS and p-stable semantics

Here, we show that the p-stable semantics extends the WFS^+ and specially the WFS semantics. These results are based on the fact that the p-stable semantics for disjunctive and in particular for normal programs is invariant under each of the transformations that define the Confluent LP-Systems CS_1 and CS_2 [5].

The following theorem indicates that the p-stable semantics is consistent with the WFS^+ semantics.

Theorem 4. *Let P be a normal program. Let $\langle T, F\rangle$ be the model of P in the WFS^+ semantics. If $\langle M, \mathcal{L}_P \setminus M\rangle$ is a p-stable model of P then $\langle T, F\rangle \leq_k \langle M, \mathcal{L}_P \setminus M\rangle$.*

Proof. Let $P \rightarrow P_1 \rightarrow \ldots \rightarrow P_n$ be the chain of reductions in the Confluent LP-System CS_2 that leads to the normalform $P_n = norm_{CS_2}(P)$. Then we have $SEM_{min}(P) \leq_k SEM_{min}(P_1) \leq_k \ldots \leq_k SEM_{min}(P_n) = WFS^+(P)$.

Since the p-stable semantics is invariant under any of the transformations that define the Confluent LP-System CS_2, then $\langle M, \mathcal{L}_P \setminus M\rangle$ is a p-stable model of each of the programs $P, P_1, \ldots, P_n$.

For any fact $a \leftarrow\ \in P$, it follows that $a \in M$ according to remark 1, hence $T \subseteq M$. By the same remark 1, $M \subset HEAD(P_n)$, therefore $\mathcal{L}_P \setminus HEAD(P_n) \subset \mathcal{L}_P \setminus M$. Thus we have $\langle T, F\rangle \leq_k \langle M, \mathcal{L}_P \setminus M\rangle$. □

Since the p-stable semantics is an extension of WFS$^+$ and WFS$^+$ is an extension of WFS, a corollary of theorem 4 indicates that the p-stable semantics is also an extension of WFS semantics.

Corollary 1. *Let P be a normal program. Let $\langle T, F\rangle$ be the model of P in the WFS semantics. If $\langle M, \mathcal{L}_P \setminus M\rangle$ is a p-stable model of P then $\langle T, F\rangle \leq_k \langle M, \mathcal{L}_P \setminus M\rangle$.*

Proof. All transformations that define the WFS semantics are contained in those that define the WFS$^+$ semantics. Let $P \to P_1 \to \ldots \to P_s$ be a chain of reductions that leads to the normalform $P_s = norm_{CS_1}(P)$. This chain can be extended to $P \to P_1 \to \ldots \to P_s \ldots \to P_n$ so that $P_n = norm_{CS_2}(P)$, where $n \geq s$.
Then we have $SEM_{min}(P_s) \leq_k SEM_{min}(P_n)$, i.e., $SEM_{CS_1}(P) \leq SEM_{CS_2}(P)$. Then the result follows from Theorem 4. □

6 Conclusions

We showed that the p-stable semantics extends the well known WFS semantics. This result is based on the fact that WFS can be induced in a natural way by a confluent LP-system, which is based on certain transformations rules; and on the fact that the p-stable semantics for normal programs is invariant under each of these transformations.

References

1. A. Avron. Natural 3-valued logic characterization and proof theory. *J. Symb. Logic*, 56(1):276–294, 1991.
2. T. J. M. Bench-Capon and P. E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10-15):619–641, 2007.
3. S. Brass, J. Dix, B. Freitag, and U. Zukowski. Transformation-based bottom-up computation of the well-founded model. *Theory Pract. Log. Program.*, 1(5):497–538, 2001.
4. J. Carballido, J. Nieves, and M. Osorio. Inferring preferred extensions by pstable semantics. *Revista Iberomericana de Inteligencia Artificial*, 13(41):38–53, 2009.
5. J. Carballido, M. Osorio, and J. Arrazola. Equivalence for the G'$_3$-stable models semantics. *Submmitted to Journal of Applied Logic*, 2009.
6. W. A. Carnielli and J. Marcos. A taxonomy of **C**-Systems. In *Paraconsistency: The Logical Way to the Inconsistent, Proceedings of the Second World Congress on Paraconsistency (WCP 2000)*, number 228 in Lecture Notes in Pure and Applied Mathematics, pages 1–94. Marcel Dekker, Inc., 2002.
7. N. da Costa. On the theory of inconsistent formal systems. *Notre Dame Journal of Formal Logic*, 15(4):497–510, 1974.
8. J. Dix. A framework for representing and characterizing semantics of logic programs. In *KR*, pages 591–602, 1992.
9. J. Dix, M. Osorio, and C. Zepeda. A General Theory of Confluent Rewriting Systems for Logic Programming and its applications. *Annals of Pure and Applied Logic*, 108(1–3):153–188, 2001.

10. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.
11. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.
12. V. Lifschitz. Foundations of logic programming. in principles of knowledge representation, pages 69-127. CSLI publications, 1996.
13. J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, second edition, 1987.
14. M. Osorio and J. L. Carballido. Brief study of G'$_3$ logic. *To appear in Journal of Applied Non-Classical Logic*, 18(4):79–103, 2009.
15. M. Osorio, J.Arrazola, and J. L. Carballido. Logical weak completions of paraconsistent logics. *Journal of Logic and Computation*, Published on line on May 9, 2008.
16. M. Osorio and J. A. Navarro. Modal logic $S5_2$ and FOUR (abstract). In *2003 Annual Meeting of the Association for Symbolic Logic*, Chicago, June 2003.
17. M. Osorio, J. A. Navarro, J. Arrazola, and V. Borja. Ground nonmonotonic modal logic S5: New results. *Journal of Logic and Computation*, 15(5):787–813, 2005.
18. M. Osorio, J. A. Navarro, J. Arrazola, and V. Borja. Logics with common weak completions. *Journal of Logic and Computation*, 16(6):867–890, 2006.
19. M. Osorio and J. C. Nieves. Pstable semantics for possibilistic logic programs. In *MICAI 2007: Advances in Artificial Intelligence, 6th Mexican International Conference on Artificial Intelligence*, number 4827 in LNAI, pages 294–304. Springer-Verlag, 2007.
20. M. Osorio and C. Zepeda. Update sequences based on minimal generalized pstable models. In *MICAI*, pages 283–293, 2007.
21. M. Osorio and C. Zepeda. Pstable theories and preferences. In *Electronic Prceedings of the 18th International Conference on Electronics, Communications, and Computers (CONIELECOMP 2008)*, March, 2008.
22. S. Pascucci and A. L. Fernandez. Syntactic transformation rules under p-stable semantics: theory and implementation. *Accepted in Revista Iberomericana de Inteligencia Artificial*, 2008.
23. D. Pearce. Stable Inference as Intuitionistic Validity. *Journal of Logic Programming*, 38:79–91, 1999.
24. T. C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing*, 9(3/4):401–424, 1991.
25. D. van Dalen. *Logic and Structure*. Springer, Berlin, second edition, 1980.
26. A. van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.
27. P. A. Veloso and W. A. Carnielli. Logics for qualitative reasoning. In *Logic, Epistemology, and the Unity of Science*.
28. C. Zepeda and J. L. Carballido. Computing of p-stable models based on seminegative normal programs with constraints. In *Proceedings of the Ninth Mexican International Conference on Computer Science (ENC 2008)*, pages 203–210, Baja California, México, October,2008.