P-stable models of Strong kernel programs

Jose Luis Carballido[1] and Claudia Zepeda[2]

[1] Universidad Politécnica de Puebla
Tercer Carril del Ejido"Serrano",
San Mateo Cuanalá s/n.,
Juan C. Bonilla, Puebla, México
jlcarballido7@gmail.com,
[2] Benemérita Universidad Atónoma de Puebla
Facultad de Ciencias de la Computación,
Puebla, Puebla, México
czepedac@gmail.com

Abstract Currently non-monotonic reasoning (NMR) is a promising approach to model features of common sense reasoning. In order to formalize NMR the research community has applied monotonic logics. The present paper furthers the study of one of the semantics useful in this formalization called *p-stable*. We introduce three different formats for normal programs: *Negative normal programs*, *Restricted negative normal programs* and *Strong kernel programs*. This forms helps to simplify the search of p-stable models of the original program. One of the main results of this paper indicates that the p-stable semantics for strong-kernel programs agrees with the stable semantics for kernel programs as defined in [2]. In this way all the applications based on stable semantics of kernel programs can also be based on p-stable semantics of strong kernel programs.
Keywords: Logic programming, p-stable semantics, stable semantics, kernel programs.

1 Introduction

Currently non-monotonic reasoning (NMR) is a promising approach to model features of common sense reasoning. In order to formalize NMR the research community has applied monotonic logics. The present paper furthers the study of one of the semantics useful in this formalization called *p-stable*.

In [7] was suggested the use of Autoepistemic Logic (AL) as an alternative formalization of NMR to avoid the problems encountered with standard modal logics. Later Gelfond and Lifschitz defined the stable model semantics [4] based on an interpretation of $\neg a$. This interpretation characterizes perfect models of stratified logic programs in terms of the extensions of the corresponding autoepistemic theory. More recently, in [1] is described how the stable semantics of disjunctive programs can be characterized in terms of AL via Gelfond's translation.

Additionally Pearce in [12] presented a characterization of the stable model semantics in terms of a collection of logics. He proved that a formula is "entailed by a disjunctive program in the stable model semantics if and only if it belongs to every intuitionistically complete and consistent extension of the program formed by adding only negated atoms". He also showed that in place of intuitionistic logic, any proper intermediate logic can be used. The strongest intermediate logic is Goedel's 3-valued logic G_3. The construction used by Pearce is called a weak completion. Based on the work of Pearce in [12], the authors of [9] express the logic G_3 in terms of Lukaciewicz L_3-logic and gave a new characterization of stable models based on Lukaciewicz L_3-logic. In [9] is also introduced a very similar 3-valued logic based in the Lukaciewicz L_3-logic: the G'_3 logic. Later, in [10] a new semantics, based on weak completions, is defined with the G'_3 logic.

In [8] the authors generalize to disjunctive programs what was done in [10]. They introduce the *p-stable semantics* for normal programs by using a transformation similar to the one used by Gelfond and Lifschits [4]. This new semantics for disjunctive programs agrees with semantics defined in terms of weak completions for the paraconsistent logics studied in [10]. The authors also present some results that give conditions under which the concepts of stable and p-stable models agree. They show that the p-stable model semantics for normal programs is powerful enough to express any problem that can be expressed with the stable model semantics for disjunctive programs. It is important to mention that 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. For example, the following program P does not have stable models.

$$a \leftarrow \neg b.$$
$$a \leftarrow b.$$
$$b \leftarrow a.$$

However the set $\{a, b\}$ could be considered the intended model for P in classical logic. Moreover, it is the p-stable model of P.

In [11] is described a schema for the implementation of p-stable semantic using two well known open source tools: Lparse and Minisat. In [11] is also presented a prototype written in Java of a tool based on that schema.

The present paper furthers the study of *p-stable* semantics. We introduce three different formats for normal programs: *Negative normal programs*, *Restricted negative normal programs* and *Strong kernel programs*. This forms helps to simplify the search of p-stable models of the original program. Besides we show that the p-stable semantics for strong kernel programs agrees with the stable semantics for kernel programs as defined in [2]. In this way all the applications based on stable semantics of kernel programs can also be based on p-stable semantics of strong kernel programs.

In [2] is introduced the definition of kernel programs and it is shown that any propositional logic program P admits an equivalent—w.r.t. stable semantics—program in a format called *kernel*. Currently there are applications based on this

kind of programs (see [2,5,3]). Moreover, kernel programs are useful since they allow us to express well known problems as the 3-coloring problem (see [5]).

This paper is structured as follows. In section 2 we introduce the general syntax of the logic programs used in this paper. We also provide the definition of stable semantics, p-stable semantics, and kernel programs. In section 3 we give the definition of the three different formats for normal programs: negative normal programs, restricted negative normal programs, and strong kernel programs. We also show the different results about the stable semantics and the p-stable semantics of programs in these formats. Finally in section 4 we present some conclusions.

2 Background

In this section we summarize some basic concepts and definitions used to understand this paper.

2.1 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*: $\wedge$, $\vee$, $\leftarrow$, $\neg$; and *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. A *literal* is either an atom a, or the negation of an atom $\neg a$. The formula $F \equiv G$ is an abbreviation for $(F \leftarrow G) \wedge (G \leftarrow F)$. 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 denoted just by: $H \leftarrow$. In the case when the head of a clause is empty, the clause is called a constraint and is denoted by: $\leftarrow B$.

A *disjunctive* clause is a clause of the form $\mathcal{H} \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^-$ where $\mathcal{H}$ is a disjunction of atoms $h_1 \vee h_2 \vee \ldots \vee h_s$, $\mathcal{B}^+$ is a conjunction of atoms $b_1 \wedge b_2 \wedge \ldots \wedge b_n$, and $\neg\mathcal{B}^-$ is a conjunction of negated atoms $\neg b_{n+1} \wedge \neg b_{n+2} \wedge \ldots \wedge \neg b_m$. $\mathcal{H}$, $\mathcal{B}^+$, and $\mathcal{B}^-$ could be empty sets of atoms. When the set $\mathcal{H}$ contains exactly one element the clause is called *normal*. 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 the program is also of that type. For instance a set of disjunctive clauses is an *disjunctive program*, a set of normal clauses is a *normal program* and so on.

Finally we give two definitions useful to understand the definitions of stable and p-stable semantics for normal programs.

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

For any program P, the positive part of P, denoted by $POS(P)$ is the program consisting exclusively of those rules in P that do not have negated literals.

2.2 Stable and P-stable semantics

From now on we assume that the reader is familiar with the notion of classical minimal model [6].

Now we give the definition of stable semantics for normal programs.

Definition 1. *[8] Let P be a normal program and let $M \subseteq \mathcal{L}_P$. Let us put $P^M = POS(RED(P, M))$ then we say that M is a* stable model *of P if M is a minimal classical model) of P^M.*

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

Definition 2. *[8] Let P be a normal program and M be a set of atoms. We say that M is a* p-stable model of P *if*

1. *M is a classical model of P (i.e. a model in classical logic), and*
2. *the conjunction of the atoms in M is a logical consequence in classical logic of $RED(P, M)$ (denoted as $RED(P, M) \models M$).*

Example 1. Let P be the normal program:

$$b \leftarrow \neg a.$$
$$a \leftarrow \neg b.$$
$$p \leftarrow \neg a.$$
$$p \leftarrow \neg p.$$

We can verify that $M_1 = \{a, p\}$ and $M_2 = \{b, p\}$ model the rules of P. From the definition of the RED transformation we find

$$RED(P, M_1) = \{b \leftarrow \neg a, \quad a \leftarrow, \quad p \leftarrow \neg a, \quad p \leftarrow \neg p\}$$

$$RED(P, M_2) = \{b \leftarrow, \quad a \leftarrow \neg b, \quad p \leftarrow, \quad p \leftarrow \neg p\}$$

And it is clear that $RED(P, M_1) \models M_1$ and $RED(P, M_2) \models M_2$. Hence M_1 and M_2 are *p-stable models* for P.

2.3 Kernel programs

In [2] the authors show that any propositional logic program P admits an equivalent—w.r.t. stable semantics—program in a format called *kernel*. They indicate that a program in kernel format is useful to studying the existence and number of stable models. Here we present the definition of kernel programs as defined in [2]. We also mention some applications of this kind of programs.

A kernel program is composed of the atoms which are undefined in the Well-founded semantics, which are those that directly affect the existence of answer sets. The body of rules is composed of negative literals only.

Definition 3. *[2] A logic program P is a* kernel program *if the following conditions hold:*

1. *P is WFS-irreducible, i.e., the well-founded model of P is empty, $WFS(P) = \langle \emptyset, \emptyset \rangle$ (see [13]);*
2. *every rule has its body composed of negative literals only;*
3. *every atom in P occurs in the body of some rule.*

It is important to note that kernel programs are useful and interesting since they allow us to express well known problems as the 3-coloring problem. In [5] you can see an stable encoding of the 3-coloring problem.

In [2] is also showed how to check consistency of kernel programs in terms of colorings of the Extended Dependency Graph (EDG) program representation. The EDG program representation has a regular and simple structure. Moreover this representation gives the first purely-syntactic and complete characterization of consistent logic programs w.r.t. stable semantics.

The authors of [5] also consider programs in kernel form. In [5] is presented a distributed version of the adjSolver algorithm for computing stable models of logic programs in kernel form. The intrinsic parallelism of the branch-and-bound structure of adjSolver is exploited to control, with a centralized architecture, the delegation of promising search subspaces to distributed handling agents. AdjSolver was designed to solve the problem of computing stable models through the search for admissible 2-colorings of the EDG representing the logic program. The adjSolver search strategy is different from the semantics-based coloring-propagation strategy adopted in literature for stable models computation by graph coloring.

The authors of [3] define various notions of coherence, that establish what relation there should be between the stable models of the original programs, and the stable models of the programs resulting from update/modification. They introduce significant sufficient conditions for coherence on the class of kernel programs. They also indicate that logic programs in kernel form allow one to distinguish the elements the program is composed of, namely cycles and handles, and thus to reason more easily about what happens when the program is modified.

3 Negative normal programs and Restricted negative normal programs

In this section we give the definition of three different formats for normal programs: *Negative normal programs*, *Restricted negative normal programs* and *Strong kernel programs*. One of the main results indicates that the stable semantics and the p-stable semantics of a restricted negative normal program coincide. We also show how the p-stable semantics of a negative normal program or a kernel program (as defined in [2]) P corresponds to the p-stable semantics of a particular program associated to P. This new program is a restricted negative normal program and it is the result of applying a transformation to the program P.

We start by introducing the definitions of negative normal programs and restricted negative normal programs. This definitions will be useful to define

negative normal programs and restricted negative normal programs with constraints.

Definition 4. *Let P be a normal program. We say that P is a* negative normal *(NN) program if it satisfies the first of the two conditions listed below, and we say that P is a* restricted negative normal *(RNN) program if it satisfies both conditions*

1. *every rule has its body composed of negative literals only;*
2. *the head of any rule does not appear in the body of the same rule.*

Let us observe that a RNN program is a NN program, and that NN, RNN and kernel programs have the common characteristic of having the body of every rule composed of negative literals only.

The following lemma will be useful to show that the stable semantics and the p-stable semantics of a RNN program coincide. Given a RNN program P, this lemma gives a characterization for a set $M \subseteq \mathcal{L}_P$, to be a p-stable model of P.

Let P be a RNN program, for any $M \subseteq \mathcal{L}_P$, we define $S_M = \{a \mid a \leftarrow \neg b_1, \wedge \ldots \wedge \neg b_m \in P,\ \{b_1, \ldots, b_m\} \cap M = \emptyset\}$.

Lemma 1. *Let P be a RNN program. If M is p-stable model of P then $M = S_M$.*

Proof. $RED(P, M) = \{a \leftarrow \neg(B^- \cap M) \mid H \leftarrow \neg B^- \in P\}$. By part 1) of the definition of p-stable model (Definition 2), it follows that $S_M \subset M$. Let us assume that $M \setminus S_M \neq \emptyset$ and take $m \in M \setminus S_M$. Let us define an interpretation: $I : \mathcal{L}_P \rightarrow \{0, 1\}$ such that $I(a) = 1$ for each $a \in \mathcal{L}_P \setminus \{m\}$ and $I(m) = 0$.

Now, using the fact that the program P has the property that the head of each rule does not appear in its body, it follows that I is a classical model for $RED(P, M)$ but it is not a model for M (Since $I(m) = 0$) this contradicts 2) in the definition of a p-stable model (Definition 2) and the lemma is proved. □

The following theorem indicates that the stable semantics and the p-stable semantics of a RNN program coincide.

Theorem 1. *Let P be a RNN program, then the stable semantics of P coincides with the p-stable semantics of P.*

Proof. If M is a p-stable model of P then $M = S_M$ according to the lemma 1, and from the definition of P^M (Definition of a stable model), $P^M = \{a \leftarrow \mid a \in S_M\}$ then we conclude that M is a minimal model of P^M and then a stable model of P.

Let us now assume that M is a stable model of P. It follows that M is a minimal model for $P^M = \{a \leftarrow \mid a \in S_M\}$. Therefore $M = S_M$. From the fact that $P^M \subset RED(P, M)$, it follows that $RED(P, M) \models M$.

We still need to show that M is a classical model of P. Let us consider a rule $a \leftarrow \neg b_1 \wedge \neg b_2 \wedge \ldots \wedge \neg b_s \in P$, such that $a \notin M$. Then there is at least an i for which $b_i \in M$ (otherwise $a \in S_M = M$), then the rule is modeled by M. Therefore M models all of the rules in P as we wanted to show. □

Let us observe that, from the definition, in a NN program the head of any rule could appear in the body of the same rule.

We shall show how the p-stable semantics of a NN program P corresponds to the p-stable semantics of a particular program associated to P. This new program is a RNN program and it is the result of applying the following transformation to P: For any normal program P, the transformed program, denoted by $TRN(P)$ is the program that results from P after deleting from the body of each rule the atom that appears as the head of the same rule. The rules that do not present this condition remain the same.

Definition 5. *For a rule r of the form $a \leftarrow \neg a \wedge \neg b_1 \wedge \ldots \wedge \neg b_n \in P$, we define the transformation TRN as follows: $TRN(r) = a \leftarrow \neg b_1 \wedge \ldots \wedge \neg b_n$. For a rule r for which $head(r) \notin body(r)$, we define $TRN(r) = r$. For a normal program P we define the transformation TRN as follows: $TRN(P) = \{TRN(r) \mid r \in P\}$.*

Here we show how the p-stable semantics of a NN program P corresponds to the p-stable semantics of the RNN program $TRN(P)$.

Theorem 2. *Let P be a NN program, then the p-stable semantics of P coincides with the p-stable semantics of the RNN program $TRN(P)$.*

Proof. Let us assume that M is a p-stable model of P and let $a \leftarrow \neg a, \neg b_1, ...\neg b_n$ be one of the rules in P with the property that the head appears in the body. Assume that $a \notin M$. Since M is a classical model for the rule, then at least for one i, $b_i \in M$, making the rule true according to M; but then it is clear that the corresponding rule in $TRN(P)$ is also modeled by M. It follows that a classical model for P is also a classical model for $TRN(P)$. Next, by hypothesis we have that $RED(P, M) \models M$. Now, it is easy to see that for each rule $r \in P$, the rule $RED(r, M)$ is a logical consequence (in classical logic) of the rule $RED(TRN(r), M)$, Therefore we conclude that $RED(TRN(P), M) \models M$.

Now, if M is a p-stable model of $TRN(P)$, according to lemma 1, M consists of those atoms for which there exists a rule $r : a \leftarrow \neg b_1 \wedge \ldots \wedge \neg b_n$ such that $b_i \notin M$ for all i.

Let us show first that M is a classical model of P. It is enough to examine the rules in $P \setminus TRN(P)$:

$$a \leftarrow \neg a \wedge \neg b_1 \wedge \ldots \wedge \neg b_n$$

In the case for which $a \in M$, there is nothing to prove. In the case for which $a \notin M$, according to the lemma 1 there must exist $b_i \in M$ for some i, and then the rule is modeled by M. So M models P.

Now, it only remains to prove that $RED(P, M) \models M$. But again, M consists of those atoms for which there is a rule, $a \leftarrow \neg b_1 \wedge \ldots \wedge \neg b_n$ and $b_i \notin M$ for all i; then $RED(P, M)$ contains the rules $a \leftarrow$, for all $a \in M$. The desired conclusion follows now. □

We also introduce another format for normal program called *Strong Kernel program*. This last format considers the three conditions of a kernel program (as

defined in [2]) and also the condition of RNN programs about that the head of any rule does not appear in the body of the same rule.

Definition 6. *Let P be a normal program. We say that P is a* strong kernel *(SK) program if the following conditions hold:*

1. *P is WFS-irreducible, i.e., the well-founded model of P is empty, $WFS(P) = \langle \emptyset, \emptyset \rangle$ (see [13]);*
2. *every rule has its body composed of negative literals only;*
3. *every atom in P occurs in the body of some rule.*
4. *the head of any rule does not appear in the body of the same rule.*

Let us observe that SK programs are also RNN programs. Then a corollary of theorem 1 indicates that the stable semantics of a SK program coincides with the p-stable semantics of the same program.

Corollary 1. *Let P be a SK program, then the stable semantics of P coincides with the p-stable semantics of P.*

Let us observe first, that a kernel program (as defined in [2]) is a NN program, and also that if we apply the transformation TRN to a kernel program, then we obtain a SK program. So, a corollary of theorem 2 indicates that the p-stable semantics of a kernel program P corresponds to the p-stable semantics of the SK program $TRN(P)$.

Corollary 2. *Let P be a kernel program, then the p-stable semantics of P coincides with the p-stable semantics of the SK program $TRN(P)$.*

As a consequence of corollary 2, all the applications based on stable semantics of kernel programs can also be based on p-stable semantics of strong kernel programs: to check consistency of kernel programs in terms of colorings of the Extended Dependency Graph program representation, as described in [2]; a distributed version of the adjSolver algorithm for computing p-stable models of logic programs in strong-kernel form, as described in [5]; and to distinguish the elements that a program is composed of, namely cycles and handles, and thus to reason more easily about what happens when the program is modified, as described in [3]. We mentioned this applications at the end of section 2 of this paper.

We remark that kernel and, in particular strong kernel programs are useful and interesting since they allow us to express well known problems as the 3-coloring problem. Next, we illustrate the p-stable kernel encoding of the 3-coloring problem over the graph $\langle V, E \rangle = \langle \{1, 2\}, \{(1, 2)\} \rangle$. Let us note that in this program, when expressed in the p-table semantics, is necessary to implement three rules for each constraint necessary in the stable semantics. In the encoding below, those three rules are written below the corresponding constraint in the stable semantics, which has been commented out. The formal and detailed description of how to write such constraints in the p-stable semantics, is presented in [8].

%Assignment of colors to vertices

$col(1, blue) \leftarrow \neg col(1, red), \neg col(1, green).$
$col(2, blue) \leftarrow \neg col(2, red), \neg col(2, green).$
$col(1, red) \leftarrow \neg col(1, blue), \neg col(1, green).$
$col(2, red) \leftarrow \neg col(2, blue), \neg col(2, green).$
$col(1, green) \leftarrow \neg col(1, blue), \neg col(1, red).$
$col(2, green) \leftarrow \neg col(2, blue), \neg col(2, red).$

% arc(1,2) cannot join 2 nodes of the same color

% $\leftarrow \neg$**col(1,red)**, $\neg$**col(2,red)**, $\neg$**col(1, blue)**, $\neg$**col(2, blue)**.
$x \leftarrow \neg y, \neg col(1, red), \neg col(2, red), \neg col(1, blue), \neg col(2, blue).$
$y \leftarrow \neg z, \neg col(1, red), \neg col(2, red), \neg col(1, blue), \neg col(2, blue).$
$z \leftarrow \neg x, \neg col(1, red), \neg col(2, red), \neg col(1, blue), \neg col(2, blue).$

% $\leftarrow \neg$**col(1,red)**, $\neg$**col(2,red)**, $\neg$**col(1,greeen)**, $\neg$**col(2,green)**.
$x \leftarrow \neg y, \neg col(1, red), \neg col(2, red), \neg col(1, green), \neg col(2, green).$
$y \leftarrow \neg z, \neg col(1, red), \neg col(2, red), \neg col(1, green), \neg col(2, green).$
$z \leftarrow \neg x, \neg col(1, red), \neg col(2, red), \neg col(1, green), \neg col(2, green).$

% $\leftarrow \neg$**col(1,green)**, $\neg$**col(2,green)**, $\neg$**col(1,blue)**, $\neg$**col(2,blue)**.
$x \leftarrow \neg y, \neg col(1, green), \neg col(2, green), \neg col(1, blue), \neg col(2, blue).$
$y \leftarrow \neg z, \neg col(1, green), \neg col(2, green), \neg col(1, blue), \neg col(2, blue).$
$z \leftarrow \neg x, \neg col(1, green), \neg col(2, green), \neg col(1, blue), \neg col(2, blue).$

There are six p-stable models: $\{col(1, red), col(2, green)\}$; $\{col(2, green), col(1, blue)\}$; $\{col(1, green), col(2, red)\}$; $\{col(1, blue), col(2, red)\}$; $\{col(1, green), col(2, blue)\}$; and $\{col(1, red), col(2, blue)\}$. Each of them indicates a coloring for the given graph.

4 Conclusion

This paper furthers the study of p-stable semantics. We introduced three different formats for normal programs: NN programs, RNN programs and SK programs. We showed that the stable semantics and the p-stable semantics of a RNN program coincide; and that the p-stable semantics of a NN program corresponds to the p-stable semantics of a particular RNN program associated to the original program.

Since the SK format considers the three conditions of a kernel program [2] and also the condition of RNN programs about that the head of any rule does not appear in the body of the same rule then, is showed that the stable semantics of a SK program coincides with the p-stable semantics of the same program and that the p-stable semantics of a kernel program corresponds to the p-stable semantics of a particular SK program.

We also indicated that all the applications based on stable semantics of kernel programs can also be based on p-stable semantics of strong kernel programs.

We remarked that kernel and, in particular strong kernel programs are useful and interesting since they allow us to express well known problems as the 3-coloring problem.

References

1. C. Baral. *Knowledge Representation, reasoning and declarative problem solving with Answer Sets.* Cambridge University Press, Cambridge, 2003.
2. S. Constantini and A. Provetti. Normal forms for answer set programming. *Journal of Theory and Practice of Logic Programming*, 5:747–760, 2005.
3. S. Costantini, B. Intrigila, and A. Provetti. Coherence of updates in answer set programming. In *Proc. of the IJCAI-2003 Workshop on Nonmonotonic Reasoning, Action and Change, NRAC03*, pages 66–72, 2003.
4. 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.
5. G. Grossi, M. Marchi, E. Pontelli, and A. Provetti. Experiments with answer set computation over parallel and distributed architectures. In *Proceedings of 4th International Workshop on Answer Set Programming (ASP2007).*, September 2007. To appear an extended version in a special issue of the Journal of Logic and Computation.
6. J. W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, second edition, 1987.
7. R. C. Moore. Autoepistemic logic. In P. Smets, E. H. Mamdani, D. Dubois, and H. Prade, editors, *Non-Standard Logics for Automated Reasoning.* Academic Press, 1988.
8. M. Osorio, J. Arrazola, and J. L. Carballido. Logical weak completions of paraconsistent logics. *To apper in the Journal of Logic and Computation*, 2008.
9. M. Osorio, V. Borja, and J. Arrazola. Three valued logic of Łukasiewicz for modeling semantics of logic programs. In *Proceedings of IBERAMIA*, number 3315 in Lecture Notes in Computer Science, pages 343–352. Springer, 2004.
10. 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.
11. S. Pascucci and A. Lopez. Implementing p-stable with simplification capabilities. *Submmited to Inteligencia Artificial, Revista Iberoamericana de I.A.*, Spain, 2008.
12. D. Pearce. Stable Inference as Intuitionistic Validity. *Logic Programming*, 38:79–91, 1999.
13. A. van Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.