Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial
ISSN: 1137-3601
revista@aepia.org
Asociación Española para la Inteligencia Artificial
España

Pascucci, Simone; López Fernández, Alejandra
Syntactic transformation rules under P-Stable semantics: theory and implementation
Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, vol. 13, núm. 41, 2009, pp. 21-37
Asociación Española para la Inteligencia Artificial
Valencia, España

Inteligencia Artificial 41(2009), 21-37
doi: 10.4114/ia.v13i41.1028

INTELIGENCIA ARTIFICIAL

http://erevista.aepia.org/

Syntactic transformation rules under P-Stable semantics: theory and implementation

Simone Pascucci[1], Alejandra López Fernández[2]

[1]Universitá di Roma "La Sapienza"
Via Salaria, 113
Roma, 00184 (Italy)
cxjepa@yahoo.it

[2]Computational Linguistics Department
Saarland University
Saarbruecken (Germany)
alfita.chaparrita@gmail.com

AbstractWe study the simplification of normal logic programs under the P-Stable semantics, with respect to the notions of equivalence, using many of the transformation rules found in literature in the context of *Answer Set Programming* (ASP). A schema for the implementation of P-Stable semantics is provided using two well known open source tools: Lparse and Minisat. Also, a prototype written in Java of a tool based on this schema is presented. We extend the work of Fernández [4] including a simplification routine in the post-parsing phase.

Keywords: Non Monotonic Reasoning, Logical Programming, P-Stable Semantics, Syntactic Trasformation.

1 Introduction

The P-Stable semantics lies in the context of Non-Monotonic reasoning and was defined by means of a fixed-point operator in terms of classical logic [2]. Many other semantics, defined in terms of weak completion by a large class of logics, were proved to be equivalent to the P-Stable semantics for normal logic programs [2]. The well-known Stable semantics (Gelfond and Lifschitz, 1998 [18]) has been proved to be related to the P-Stable semantics, in fact given a normal logic program P, let $MM(P)$, $S(P)$ and $PS(P)$ denote respectively the set of all the minimal models of P, the set of all the Stable models of P and the set of all the P-Stable models of P, we have that $S(P) \subseteq PS(P) \subseteq MM(P)$ [2]. For example, let P be the following normal program

$$a \leftarrow not\ b$$

we have $MM(P) = \{\{a\}, \{b\}\}$, $PS(P) = \{\{a\}\}$, $S(P) = \{\{a\}\}$. Such characterization guided us in the development of the implementation schema, giving the hint on how to generate a procedure that could find the P-Stable models of a given normal program. In the context of finding quick procedures to compute P-Stable models we study in this paper many of the syntactic transformation rules found in the context of ASP [8, 9], giving results about their equivalence preserving properties under P-Stable semantics. Moreover, many syntactic transformations are not only valuable in terms of a fast computation of models, but they also provide a way to simplify programs that are used as base for larger ones, this kind of transformations are said to preserve strong equivalence (see next sections). It remains to mention

ISSN: 1988-3064(on-line)

that there are applications, such as in the theory of argumentation, in which a behavior closer to classical logic is needed [17]. In such cases the P-Stable semantics provides a solution since it can be expressed using only classical logic [17].
We present a schema for the implementation of a tool that computes the P-Stable models of a given normal logic program extending the work of Fernández [4] with the inclusion of a simplification routine in the post-parsing phase. A prototype written in Java implementing such a schema can be obtained at `http://cxjepa.googlepages.com`.
This paper is structured in the following way. In the Section 2 we resume the basic theory underlying the P-Stable semantics. In Section 3 we give the definitions and properties of the syntactic transformations that can be applied to normal logic programs. In Section 4 we expose such a schema presenting also a small prototype written in Java. Finally, in Section 5 we make our final considerations giving hints and directions about future works and improvements to the tool.

2 Basic Theory

In this section, the notation used in the rest of this paper as well as some fundamental results about P-Stable semantics are presented.

2.1 The Framework

Most of the definitions and results given here are taken "as they are" from the works of Baral [1] and Osorio et al. [2]. A more interested reader can refer to these documents to have a deeper vision of the subject. The language which we are dealing with is known to be a *declarative* one, meaning, informally, that we are able to express *what* the solution of the problem should look like instead of giving a procedure that says *how* to compute it. For both lexical and syntactic characteristics refer to Lparse user manual, [5], and Baral [1] Section 1.2. We give now the definition of the main constituents of our language:

Definition 2.1. A *term* is defined as follows:

1. A variable is a term.
2. A constant is a term.

An *atom* is of the form $p(t_1, \ldots, t_n)$ where p is a predicate symbol and every t_i is a term, for $i = 1 \ldots n$. A *naf-literal* is either an atom or an atom proceeded by the symbol **not** .

A term is said to be *ground* if no variable occurs in it. Given an atom $p(t_1, \ldots, t_n)$, if every t_i is ground, for $i = 1 \ldots n$, then the atom is said to be ground.

Remark 2.2. In this paper the so called "classical negation/strong negation" of ASP is not considered.

Definition 2.3. A *normal logic program* is a collection of *rules*, or *clauses*, of the form:

$$a_0 \leftarrow a_1, \ldots, a_m, \ \textbf{not}\ a_{m+1}, \ldots, \ \textbf{not}\ a_{m+n}.$$

where each a_i is an atom.
The parts on the left and on the right of '$\leftarrow$' are called the *head* and the *body* of the rule, respectively. A rule with an empty body is called a *fact*. Given a rule r, we define the sets $H(r) = \{a_0\}$, $B^+(r) = \{a_1, \ldots, a_m\}$, $B^-(r) = \{a_{m+1}, \ldots, a_{m+n}\}$, $B(r) = B^+(r) \cup B^-(r)$.

Note that a fact ($a_0 \leftarrow .$) is often denoted by only his head ($a_0.$). The notation just introduced follows the tradition of logic programming to define a propositional language built from an enumerable set $\mathcal{L}$ of atoms, the binary connectives $\wedge$ (conjunction) and $\rightarrow$ (implication) and the unary connective $\neg$ (negation). Given a set of atoms $M = \{a_1, \ldots, a_m\}$, we define $\neg M = \{\neg a_1, \ldots, \neg a_m\}$. In this paper a rule r as in Definition 2.3 stands for a formula of the language of the type

$$\bigwedge \left(B^+(r) \cup \neg B^-(r)\right) \rightarrow H(r)$$

while a normal program is just a set of formulas or a *theory*.

The *Herbrand universe* of a logic program is the set of all ground terms that can be constructed using constants in that program. The *Herbrand base* of a logic program is the set of all ground atoms that can be constructed using the predicates of the program and the terms of the Herbrand universe. An *instance* of an atom, a literal or a rule is constructed by replacing all variables in it by ground terms. The *Herbrand instantiation* is the set of all ground instances of the rules that may be constructed using terms in the Herbrand universe of P. The *signature* of a ground program P, notation $\mathcal{L}_P$, is the set of all ground atoms that appear in P.

The concepts related to the grounding process bypass the scope of this paper and for that reason are omitted. In the rest of the paper we are going only to consider ground finite programs.

2.2 Semantics

Before talking about the P-Stable semantics, we briefly resume some fundamental concepts of classical logic which are strictly related to the former one. For an introduction about terminology and concepts of classical logic refer for example to Ben-Ari [3]. We consider a logic X as a set of formulas that is closed under modus ponens and is closed under substitution. The elements of a logic X are called theorems and the notation $\vdash_X A$ is used to state that the formula A is a theorem of X.

Definition 2.4 (model)**.** A set of formulas $U = \{A_1, \ldots, A_n\}$ is *satisfiable* iff there exists an interpretation v such that $v(A_1) = \ldots = v(A_n) = T$, where T stands for the constant true. The satisfying interpretation is called a *model* of U.

Note that from this point on a model will be referred as the set of the atoms that evaluate true in the interpretation.

Definition 2.5 (minimal model)**.** A set of atoms M is a *minimal model* of a program P (or set of formulas U, as previously said) if M is a classical model of P and is minimal (with respect to set inclusion) among other classical models of P.

Definition 2.6 (logical consequence)**.** Let U be a set of formulas, A be a formula and X be any logic. A is a *logical consequence* of U in the logic X iff $\vdash_X (F_1 \wedge \ldots \wedge F_n) \rightarrow A$ for some formulas $F_i \in U$. Notation: $U \vdash_X A$.

Remark 2.7. Following the notation of Osorio et al. [2], we extend the above definition including the notion of logical implication between two theories T and U in any logic X using $T \vdash_X U$ to state the fact that $T \vdash_X F$ for all formulas $F \in U$. If M is a set of atoms we write $T \Vdash_X M$ when $T \vdash_X M$ and M is a classical two-valued model of T.

The definition of a P-Stable model is now given with some fundamental results that have been used in the implementation schema (refer to the work of Osorio [2] Section 4.3).

Definition 2.8 (Reduction)**.** Let P be a normal program and M a set of atoms. We define *RED(P,M)* as the following set of clauses

$$\{H(r) \leftarrow \mathcal{B}^+(r),\ not\ (\mathcal{B}^-(r) \cap M) | r \in P\}$$

Definition 2.9 (P-Stable model)**.** Let P be a normal program, and M a set of atoms. We say that M is a *P-Stable model* of P if $RED(P, M) \Vdash_C M$.

Note that the subscript C denotes the fact that we are talking about classical logic semantics.

Example 2.10. Let P be the program:

$$\begin{aligned} P: \quad & a \leftarrow not\ b. && (1)\\ & b \leftarrow not\ a. && (2)\\ & p \leftarrow not\ p. && (3)\\ & p \leftarrow a. && (4) \end{aligned}$$

then, given the set of atoms $M = \{p, a\}$, we have

$$\begin{aligned} RED(P,M): \quad & a. && (5)\\ & b \leftarrow not\ a. && (6)\\ & p \leftarrow not\ p. && (7)\\ & p \leftarrow a. && (8) \end{aligned}$$

M is a P-Stable model of P. To see that, first note that M is a model of $RED(P, M)$. Moreover, $RED(P, M) \vdash_C M$ because does not exist any other model M' of $RED(P, M)$ such that $M \nsubseteq M'$. This follows from the syntactic structure of rules (5) and (8).

To conclude the section, we reproduce a theorem that state the relation between minimal and P-Stable models.

Theorem 2.11 (Osorio et al. [2])**.** *Let P be any theory, and M a set of atoms. If M is a P-Stable model of P then M is a minimal model of P.*

Example 2.12. Let P be the normal logic program as defined in Example 2.10, $M = \{p, a\}$ and $M' = \{p, a, b\}$. We have that $RED(P, M') = RED(P, M) \setminus \{a.\} \cup \{a \leftarrow not\ b.\} = P$. M, which is a P-Stable model of P, is also a minimal model of P. M' is a model (not minimal) of P, then it is also a model of $RED(P, M')$ but we have that $RED(P, M') \nvdash_C M'$. So, M' is not a P-Stable model of P.

3 Transformations rules

In this section we study some transformations already defined in logic programming literature, giving results about their applicability to normal logic programs under P-Stable semantics. We know from Osorio et al. [9] that **SUB**, **TAUT**, **RED**$^-$ and **SUC** preserves strong equivalence under Stable semantics, while **RED**$^+$ and **GPPE** preserves equivalence only. We briefly resume some definitions and properties of logic programs under P-Stable semantics such as the notions of equivalence and their relation with the logic G'_3.

Definition 3.1. Let P, P' be two normal logic programs. We say that:

(a) P is *equivalent* to P', notation $P \equiv_e P'$, if P and P' have the same set of P-Stable models.

(b) P is *uniformly equivalent* to P', notation $P \equiv_u P'$, if for any set of atoms M, $P \cup M$ and $P' \cup M$ are equivalent.

(c) P is *strongly equivalent* to P', notation $P \equiv_s P'$, if for any normal logic program P'', $P \cup P''$ and $P' \cup P''$ are equivalent.

Note that from the definition follows that $(c) \Rightarrow (b) \Rightarrow (a)$. The G'_3-logic is a logic system which has been used to introduce a new semantics for non-monotonic reasoning and which has been proved to be equivalent to the P-Stable model semantics for normal program [2]. One of the main results of Osorio et al. [11] is that for any two arbitrary logic programs P and P' if $P \equiv_{G'_3} P'$ then P and P' are strongly equivalent under P-Stable semantics. For an axiomatization of G'_3 refer to the work of Osorio [15]. Basic transformation rules are defined in terms of binary relations on the set $\mathcal{P}$ of all programs [9]. We consider a basic transformation rule B as a syntactic operator $B : \mathcal{P} \to \mathcal{P}$ which is identified with the identity operator when the conditions for its applicability are not satisfied.

3.1 Transformation rules preserving equivalence

Definition 3.2 (sub-implication)**.** A rule r' is a *sub-implication* of another rule r such that $r \neq r'$, symbolically $r \blacktriangleleft r'$, iff $H(r) = H(r')$, $B^+(r) \subseteq B^+(r')$, $B^-(r) \subseteq B^-(r')$.

Note that **SUB** is also known in the literature of automated reasoning and artificial intelligence as *subsumption*.

Definition 3.3 (SUB)**.** Let P be a normal logic program and $r, r' \in P$ such that $r \blacktriangleleft r'$. Then, replace P by $P \setminus \{r'\}$.

Example 3.4. Let P be the program:

$$\begin{aligned} P: \quad & a \leftarrow b, \mathit{not}\ b. && (1)\\ & a \leftarrow b, e, \mathit{not}\ b, \mathit{not}\ a. && (2) \end{aligned}$$

we can apply **SUB** to clauses (1) and (2) to obtain the following reduced program P':

$$P': \quad a \leftarrow b, \mathit{not}\ b. \qquad (1)$$

SUB preserves strong equivalence also under P-Stable semantics, as shown in the next proposition.

Proposition 3.5. *Let P be a normal logic program. Then $P \equiv_s SUB(P)$.*

Proof. Let $P \setminus \{r'\}$ be denoted by P', $B^+(r') \setminus B^+(r)$ by Q^+, $B^+(r') \setminus B^+(r)$ by Q^- . We have to show that (i) for all clauses F in P, $P' \vdash_{G'_3} F$, and (ii) for all clauses G in P', $P \vdash_{G'_3} G$. (i) The result follows directly from the proof for $P' \vdash_{G'_3} \bigwedge(B^+(r') \cup \neg\, B^-(r')) \rightarrow H(r')$:

$$\begin{aligned} & P', Q^+, \neg\, Q^- && \vdash_{G'_3}\\ & \textstyle\bigwedge(B^+(r) \cup \neg\, B^-(r)) \rightarrow H(r) && \text{Asmpt (1)}\\ & P', Q^+ && \vdash_{G'_3}\\ & (\textstyle\bigwedge(B^+(r') \cup \neg\, B^-(r)) \rightarrow H(r)) && \text{Ded 1 (2)}\\ & P' && \vdash_{G'_3}\\ & (\textstyle\bigwedge(B^+(r') \cup \neg\, B^-(r')) \rightarrow H(r)) && \text{Ded 2 (3)} \end{aligned}$$

(ii) For all G in P' we have that G is also in P. □

Definition 3.6 (TAUT)**.** Let P be a normal logic program and $r \in P$ a rule such that $H(r) \in B^+(r)$. Then, replace P by $P \setminus \{r\}$.

Example 3.7. Let P be the program:

$$\begin{aligned} P: \quad & a \leftarrow a, \mathit{not}\ b. && (1)\\ & b \leftarrow \mathit{not}\ a. && (2) \end{aligned}$$

we can apply **TAUT** to clause (1) to obtain the following reduced program P':

$$P': \quad b \leftarrow \mathit{not}\ a. \qquad (1)$$

Proposition 3.8. *Let P be a normal logic program. Then, $P \equiv_s TAUT(P)$.*

Proof. The proof is similar to the one of Proposition 3.5 considering that $\vdash_{G'_3} a \rightarrow a$. □

Definition 3.9 (SUC)**.** Let P be a normal logic program, $r, r' \in P$ such that $H(r') \in B^+(r)$, $B(r') = \emptyset$. Then, replace P by $P \setminus \{r\} \cup \{r''\}$, where r'' is

$$H(r) \leftarrow (B^+(r) \setminus H(r')), \mathit{not}\ B^-(r).$$

The transformation **SUC** is also well known in literature as *unit propagation*.

Example 3.10. Let P be the program:

$$\begin{aligned} P: \quad & c \leftarrow \mathit{not}\ b. && (1)\\ & b \leftarrow a,\ \mathit{not}\ c. && (2)\\ & a. && (3) \end{aligned}$$

we can apply **SUC** to clause (2) to obtain the following reduced program P':

$$\begin{aligned} P': \quad & c \leftarrow \mathit{not}\ b. && (1)\\ & b \leftarrow \mathit{not}\ c. && (2)\\ & a. && (3) \end{aligned}$$

Proposition 3.11. *Let P be a normal logic program. Then, $P \equiv_s SUC(P)$.*

Proof. $(P \vdash_{G'_3} \{P \setminus \{r\}\} \cup \{r''\})$.

$$\begin{aligned} & P \vdash_{G'_3} \bigwedge(B(r) \cup \{a\}) \rightarrow H(r) && \text{Asmpt} && (1)\\ & P \vdash_{G'_3} a && \text{Asmpt} && (2)\\ & P \vdash_{G'_3} a \rightarrow (\bigwedge(B(r)) \rightarrow H(r)) && \text{Equiv 6} && (3)\\ & P \vdash_{G'_3} \bigwedge(B(r)) \rightarrow H(r) && \text{MP 7,8} && (4) \end{aligned}$$

$(\{P \setminus \{r\}\} \cup \{r''\} \vdash_{G'_3} P)$. Easy derivation using the axiom $a \rightarrow (b \rightarrow a)$ and the equivalence $(a \rightarrow (b \rightarrow c)) \rightarrow (a \wedge b \rightarrow c)$. □

The notion of *s-implication* has been given from Wang and Zhou in [10] for disjunctive normal programs. The analogous version for normal logic program is given in the next definition.

Definition 3.12 (s-implication)**.** A rule r' is an *s-implication* of another rule r such that $r \neq r'$, symbolically $r \lhd r'$, iff there exists an atom $a \in B^-(r')$ such that

(i) $H(r) = \{a\}$

(ii) $B^-(r) \subseteq B^-(r') \setminus \{a\}$

(iii) $B^+(r) \subseteq B^+(r')$

Definition 3.13 (SS-IMP)**.** Let P be a normal logic program, $r, r', r'' \in P$ such that $r \lhd r'$, $H(r) = a$ as defined in point (i) of Definition 3.12, $H(r'') = H(r')$, $B^+(r'') = \{a\}$ and $B^-(r'') = \{a\}$. Then, replace P by $P \setminus \{r'\}$.

This is a stronger version of **S-IMP**, see next subsection, requiring the presence of an extra rule r'' in the program P. Such extra requirement assures that the application of the transformation **SS-IMP** preserves strong equivalence under P-Stable semantics.

Example 3.14. We give an example of a normal program P which structure satisfies the applicability of **SS-IMP**. Let P be the program of example 3.37. Then, let $P' = P \cup \{a \leftarrow b,\ \mathit{not}\ b\}$. We have that P' and *SS-IMP(P)* have the same set of P-Stable models $\{\{b, e\}, \{a\}\}$.

Proposition 3.15. *Let P be a normal logic program. Then, $P \equiv_s$SS-IMP(P).*

Proof. Let P_1 be any normal program and M any set of atoms. We start by proving *RED(SS-IMP(P)*$\cup P_1$, *M)*$\vdash_C$ *RED*$(P \cup P_1, M)$. Without loss of generality, let us assume $B^-(r) = B^-(r') \setminus \{a\}$, $B^+(r) = B^+(r')$

and let $r'' = RED(r', M)$, $b = H(r')$, $C = \left(\bigwedge_{c \in B^+(r)} c\right)$, $D = \left(\bigwedge_{d \in B^-(r'')} \neg d\right)$. We first consider the case when $a \notin M$,

$$
\begin{array}{lr}
(a \wedge \neg a) \rightarrow b, (C \wedge D) \rightarrow a, C \wedge D & \vdash_{G'_3} \\
(C \wedge D) \rightarrow a & \text{Asmpt (10)} \\
(a \wedge \neg a) \rightarrow b, (C \wedge D) \rightarrow a, C \wedge D & \vdash_{G'_3} \\
C \wedge D & \text{Asmpt (11)} \\
(a \wedge \neg a) \rightarrow b, (C \wedge D) \rightarrow a, C \wedge D & \vdash_{G'_3} \\
a & \text{MP 11, 10 (12)} \\
(a \wedge \neg a) \rightarrow b, (C \wedge D) \rightarrow a, C \wedge D & \vdash_{G'_3} \\
(a \wedge \neg a) \rightarrow b & \text{Asmpt (13)} \\
(a \wedge \neg a) \rightarrow b, (C \wedge D) \rightarrow a, C \wedge D & \vdash_{G'_3} \\
a \rightarrow (\neg a \rightarrow b) & \text{Equiv 13 (14)} \\
(a \wedge \neg a) \rightarrow b, (C \wedge D) \rightarrow a, C \wedge D & \vdash_{G'_3} \\
\neg a \rightarrow b & \text{MP 12, 14 (15)} \\
(a \wedge \neg a) \rightarrow b, (C \wedge D) \rightarrow a & \vdash_{G'_3} \\
(C \wedge D) \rightarrow (\neg a \rightarrow b) & \text{Ded. 15 (16)} \\
(a \wedge \neg a) \rightarrow b, (C \wedge D) \rightarrow a & \vdash_{G'_3} \\
(C \wedge D \wedge \neg a) \rightarrow b & \text{Equiv 16 (17)}
\end{array}
$$

By the definition of C, D and b follows the result. The case when $a \in M$ is analogous. For the converse, just note that every rule of *SS-IMP(P)* is also in P. □

Definition 3.16 (RED$^+$)**.** Let P be a normal logic program, $r \in P$ a rule and a an atom such that:

(i) $a \in B^-(r)$,

(ii) $\nexists r' \in P$ such that $\{a\} = H(r')$.

Then, replace P by $P \setminus \{r\} \cup \{r''\}$, where r'' is $H(r) \leftarrow B^+(r)$, *not* $(B^-(r) \setminus \{a\})$.

Note that the application of **RED**$^+$ could generate the conditions satisfying the applicability of some other transformations. The transformation **RED**$^+$ does not preserve uniform equivalence as shown in the next example.

Example 3.17. Let P be the following program,

$$
\begin{array}{lll}
P: & a \leftarrow b,\ e,\ not\ b,\ not\ c. & (1) \\
& b \leftarrow e,\ not\ c & (2) \\
& b \leftarrow not\ a. & (3) \\
& e \leftarrow b,\ not\ f. & (4)
\end{array}
$$

P' is the program obtained from P after three applications of **RED**$^+$, respectively to rules (1), (2), (4):

$$
\begin{array}{lll}
P': & a \leftarrow b,\ e,\ not\ b. & (1) \\
& b \leftarrow e & (2) \\
& b \leftarrow not\ a. & (3) \\
& e \leftarrow b. & (4)
\end{array}
$$

P and P' have the same P-Stable models $\{e, b\}$, $\{a\}$. However, $P \cup \{f.\}$ and $P' \cup \{f.\}$ have different P-Stable models, respectively $\{f, b\}$ and $\{f, b, e\}, \{f, a\}$.

We show now that $\mathbf{RED}^+$ preserves equivalence.

Lemma 3.18. *Let P be a normal logic program, $r \in P$ and $a \in B(r)$ such that does not exist a rule $r' \in P : H(r') = \{a\}$. Then, for any set of atoms M, $RED(P, M) \nvdash a$.*

Proof. Let

$$M' = \bigcup_{r \in RED(P,M)} H(r).$$

The result follows from the considerations that $a \notin M'$ and M' is a classical model of $RED(P, M)$. □

Proposition 3.19. *Let P be a normal logic program. Then, $P \equiv_e RED^+(P)$.*

Proof. Let M be a P-Stable model of P, $r \in P$ and $a \in B(r)$ such that does not exist a rule $r' \in P : H(r') = a$. By Lemma 3.18 we have that $a \notin M$, then, M is a classical two-valued model also of $RED^+(P)$. By Definition 2.8, we have that $RED(P, M) = RED(RED^+(P), M)$. The converse part is proved by a symmetric argument. □

NAF is the abbreviation for Negation as Failure, a transformation that completes the preceding $\mathbf{RED}^+$:

Definition 3.20 (NAF)**.** Let P be a normal logic program, $r \in P$ a rule and a an atom such that:

(i) $a \in B^+(r)$,

(ii) $\nexists r' \in P$ such that $\{a\} = H(r')$.

Then, replace P by $P \setminus \{r\}$.

We remark that **NAF** is also known under the name of *Failure* [19]. **NAF** does not preserve uniform equivalence as shown in the next example.

Example 3.21. Let P be the following program,

$$\begin{aligned} P: \quad & a \leftarrow \mathit{not}\ b. && (1)\\ & b \leftarrow \mathit{not}\ a. && (2)\\ & a \leftarrow e. && (3) \end{aligned}$$

P' is the program obtained from P after three applications of **NAF** to rule (3):

$$\begin{aligned} P': \quad & a \leftarrow \mathit{not}\ b. && (1)\\ & b \leftarrow \mathit{not}\ a. && (2) \end{aligned}$$

P and P' have the same P-Stable models $\{b\}$, $\{a\}$. However, $P \cup \{e.\}$ and $P' \cup \{e.\}$ have different P-Stable models, respectively $\{e, a\}$ and $\{e, b\}$, $\{e, a\}$.

NAF preserves equivalence, as stated in Proposition 3.23.

Lemma 3.22 ([7])**.** *Let P be any theory and M a set of atoms. $P \cup \neg\widetilde{M} \Vdash_C M$ iff M is a minimal model of P.*

Proposition 3.23. *Let P be a normal logic program. Then, $P \equiv_e NAF(P)$.*

Proof. Let M_2 be a P-Stable model of $NAF(P)$, $r \in P$ and $a \in B^+(r)$ such that points (i) and (ii) of Definition 3.20 are satisfied. By Lemma 3.18, $a \notin M_2$ then M_2 is a classical model of P. Moreover, $RED(P, M_2) \vdash_C RED(NAF(P), M_2)$, then by transitivity we have $RED(P, M_2) \vdash_C M_2$. Conversely, let M_1 be a P-Stable model of P. By Definition 2.9, M_1 is a classical two valued model of P, being *NAF(P)* composed by a subset of the rules present in P we have that M_1 is also a classical model of *NAF(P)*. Also, let $\widetilde{M_1} = \mathcal{L}_P \setminus M_1$, by Theorem 2.11 we have that $RED(P, M_1) \vdash_C M_1$ implies $RED(P, M_1) \cup \neg\widetilde{M_1} \vdash_C M_1$. By Lemma 3.18, $a \in \widetilde{M_1}$ hence $\neg a \in RED(P, M_1) \cup \neg\widetilde{M_1}$ and $\neg a \in RED(NAF(P), M_1) \cup \neg\widetilde{M_1}$. For any

atom b, $RED(NAF(P), M_1) \cup \neg\widetilde{M_1} \vdash_C a \rightarrow b$, then $RED(NAF(P), M_1) \cup \neg\widetilde{M_1} \equiv_C RED(P, M_1) \cup \neg\widetilde{M_1}$. By Lemma 3.22, M_1 is a minimal model of $RED(NAF(P), M_1)$, to conclude the proof, let assume by contradiction that $RED(NAF(P), M_1) \nvdash_C M_1$, this implies the existence of a minimal model M_2 of $RED(NAF(P), M_1)$ such that $M_1 \cap M_2 \neq M_1$. M_2 is also a model of $RED(P, M_1)$ then follows the contradiction $RED(P, M_1) \nvdash_C M_1$. □

RED$^-$, despite to his name, can be considered the completion of the transformation **SUC**:

Definition 3.24 (RED$^-$)**.** Let P be a normal logic program, $r, r' \in P$ such that $H(r') \subseteq B^-(r)$, $B(r') = \emptyset$. Then, replace P by $P \setminus \{r\}$.

Example 3.25. Let P be the program:

$$P: \quad a \leftarrow not\ b. \qquad (1)$$
$$b. \qquad (2)$$

we can apply **RED$^-$** to clause (1) to obtain the following reduced program P':

$$P': \quad b. \qquad (1)$$

Proposition 3.26. *Let P be a normal logic program. Then, $P \equiv_s RED^-(P)$.*

Note that in the proof we cannot use the fact that for any two arbitrary logic programs P and P' if $P \equiv_{G'_3} P'$ then P and P' are strongly equivalent under P-Stable semantics. This because given a normal logic program P which satisfies the applicability of **RED$^-$**, we are not able to prove in G'_3 a formula of the type $\neg a \rightarrow b$ having a between the hypothesis. This would require the presence of the axiom $a \rightarrow (\neg a \rightarrow b)$ which is one of the axiom we can add to G'_3 to obtain classical logic [15].

Proof. Let P_1 be any normal logic program and $r, r' \in P$ as in Definition 3.24, we must show that $P \cup P_1 \equiv_e RED^-(P) \cup P_1$. Note that a set of atoms M is a model of $P \cup P_1$ iff M is a model of $RED^-(P) \cup P_1$. This because any such model M must contain $H(r')$ and from this consideration follows that r is satisfied for any other truth assignment to its remaining atoms. Let consider now $RED(RED^-(P), M)$ and $RED(P, M)$. $RED(P, M) \vdash_C RED(RED^-(P), M)$ because every formula in $RED(RED^-(P), M)$ is also in $RED(P, M)$. Moreover, we can now use the axiom $a \rightarrow (\neg a \rightarrow b)$ to easily prove r from the set of hypothesis $RED(RED^-(P), M)$. It follows that $RED(P, M) \equiv_C RED(RED^-(P), M)$. □

Definition 3.27 (EQUIV)**.** Let P be a normal logic program, $r \in P$ a rule such that $H(r) \in B^-(r)$. Then, replace P by $P \setminus \{r\} \cup \{r'\}$, where

$$r' = H(r) \leftarrow B(r),\ not\ (B^-(r) \setminus H(r)).$$

Example 3.28. Let P be the program:

$$P: \quad a \leftarrow not\ b,\ not\ a. \qquad (1)$$
$$b \leftarrow not\ a. \qquad (2)$$

we can apply **EQUIV** to clause (1) to obtain the following reduced program P':

$$P': \quad a \leftarrow not\ b. \qquad (1)$$
$$b \leftarrow not\ a. \qquad (2)$$

Proposition 3.29. *Let P be a normal logic program, $r \in P$ a rule such that $H(r) \in B^-(r)$. Then, $P \equiv_s EQUIV(P)$.*

Proof. It follows from a simple check of the truth tables of connectives in G'_3 that

$$\bigwedge(B^+(r) \cup \neg\,(B^-(r))) \rightarrow H(r) \equiv_{G'_3}$$
$$\bigwedge(B^+(r) \cup \neg\,(B^-(r) \setminus H(r))) \rightarrow H(r)$$

□

The transformation **LOOP** has been shown to preserve equivalence under Stable semantics [9]. Let $definite(P)$ denote the definite program obtained from a normal logic program P removing every negative literal in P. It is a well know result that definite programs have a unique minimal model, refer for example to the work of Baral [1]. By $MM(Definite(P))$ we mean the unique minimal model of $Definite(P)$.

Definition 3.30 (LOOP)**.** Let $unf(P) = \mathcal{L}_P \setminus MM(Definite(P))$. Then, we define $LOOP(P) = \{r \in P \mid B^+(r) \cap unf(P) = \emptyset\}$.

LOOP does not preserve uniform equivalence as shown in the next example, a proof for equivalence can instead be found in the work of Carballido [16].

Example 3.31. Let P be the program:

$$\begin{array}{lll} P: & a \leftarrow e, \mathit{not}\ b. & (1)\\ & b \leftarrow c. & (2)\\ & e \leftarrow \mathit{not}\ b. & (3)\\ & c \leftarrow d. & (4)\\ & d \leftarrow c. & (5) \end{array}$$

we have that $MM(Definite(P)) = \{a, e\}$, $unf(P) = \mathcal{L}_P \setminus MM(Definite(P)) = \{a, b, c, d, e\} \setminus \{a, e\} = \{c, d, b\}$, $LOOP(P) = \{(1), (3)\}$. P and $LOOP(P)$ have just one P-Stable model $\{e, a\}$ but $P \cup \{c.\}$ and $LOOP(P) \cup \{c.\}$ have different P-Stable models, respectively $\{c, d, b\}$ and $\{c, e, a\}$.

3.2 Transformation rules not preserving equivalence

We grouped in this subsection many of the transformation rules found in literature that do not preserve any type of equivalence, the uninterested reader can safely skip to the next section.

Definition 3.32 (CONTRA)**.** Let P be a normal logic program and $r \in P$ such that $B^+(r) \cap B^-(r) \neq \emptyset$. Then, replace P by $P \setminus \{r\}$.

CONTRA does not preserve any type of equivalence, as shown in the next example.

Example 3.33.

$$\begin{array}{lll} P: & b \leftarrow a, \mathit{not}\ a. & (1)\\ & a \leftarrow \mathit{not}\ b. & (2) \end{array}$$

P has two P-Stable models $\{b\}, \{a\}$ while $P \setminus \{(1)\}$ has just one P-Stable model $\{a\}$.

Definition 3.34 (GPPE)**.** Let P be a normal logic program, $r \in P$ a rule, $a \in B^+(r)$ an atom, $G_a \neq \emptyset$ for $G_a = \{r' \in P \mid \{a\} = H(r')\}$. Then, replace P by $P \setminus \{r\} \cup \{G_a'^{\ddagger}\}$, where

$$\begin{array}{l} G_a'^{\ddagger} = \{H(r) \leftarrow (B^+(r) \setminus \{a\}) \cup B^+(r'),\\ \qquad \mathit{not}\ (B^-(r) \cup B^-(r')) \mid r' \in G_a\}. \end{array}$$

In P-Stable semantics, **GPPE** does not preserve equivalence as shown in the next example.

Example 3.35. Let P be the following normal program:

$$\begin{array}{lll} P: & a \leftarrow c. & (1)\\ & c \leftarrow \mathit{not}\ b. & (2)\\ & c. & (3)\\ & b. & (4) \end{array}$$

P has one P-Stable model $\{b, c, a\}$. After the application of **GPPE** to rule (1), considering the atom $c \in B^+((1))$, we obtain the following program P'

$$\begin{aligned} P': \quad & a \leftarrow \mathit{not}\ b. && (1)\\ & c \leftarrow \mathit{not}\ b. && (2)\\ & c. && (3)\\ & b. && (4) \end{aligned}$$

which has one P-Stable model $\{c, b\}$.

With regard to the weak version of **GPPE**, **WGPPE** [8], when applied to normal logic programs it just adds redundant rules and for that reason is not considered here. We give now the definition of **S-IMP** which has been reported to preserve strong equivalence in Stable semantics by Eiter [8].

Definition 3.36 (S-IMP)**.** Let P be a normal logic program and $r, r' \in P$ such that $r \lhd r'$. Then, replace P by $P \setminus \{r'\}$.

S-IMP does not preserve equivalence as we can see from the next example.

Example 3.37. Let P be the program:

$$\begin{aligned} P: \quad & a \leftarrow e, \mathit{not}\ b. && (1)\\ & b \leftarrow e. && (2)\\ & b \leftarrow \mathit{not}\ a. && (3)\\ & e \leftarrow b. && (4) \end{aligned}$$

P has the two P-Stable models $\{e, b\}$, $\{a\}$. After the removal of (1) due to the application of **S-IMP**, note that (2) $\lhd$ (1), the obtained reduced program has just one P-Stable model $\{e, b\}$.

We summarize the results of this section in table 1.

4 Implementation schema

We present the schema directly referring to the prototype written in java we implemented. The source code is divided into five main packages:

- main.
- dataStructures.
- fileManagement.
- transformations.
- pstable.

We just mention that our prototype makes intensive use of two third party tools, Lparse and Minisat, respectively available at `http://minisat.se` and `http://www.tcs.hut.fi/Software/smodels`. Lparse is a front-end that generates a variable-free simple logic program. The reason of this choice is that in our opinion Lparse is the most feature-rich of the different parsers and front ends. We decided to use MiniSat because of its high efficiency. How reported at MiniSat web page, this SAT solver is the winner of all the industrial categories of the SAT 2005 competition. The reader interested to the technical background can refer to their respective user manuals. Also, a brief survey of their usage can be found in [4] and at the prototype web page (`http://cxjepa.googlepages.com`).

4.1 main

The main package contains the main method. It implements the tool external interface, which can be used in any application requiring the computation of the P-Stable models of a given normal logic program.

	SUB	TAUT	SUC	SS-IMP	CONTRA	RED$^+$	NAF	RED$^-$	GPPE	EQUIV	LOOP
$\equiv_e$	yes	yes	yes	yes	no	yes	yes	yes	no	yes	yes
$\equiv_u$	yes	yes	yes	yes	no	no	no	yes	no	yes	no
$\equiv_s$	yes	yes	yes	yes	no	no	no	yes	no	yes	no

Table 1: Summary of the transformation rules. For each notion of equivalence $\equiv_t$, for $t \in \{e, u, s\}$, and each transformation rule R, it is reported if, given a normal logic program P, $P \equiv_t R(P)$.

input : A file *fileName* containing the description of a normal logic program, a integer n representing the number of P-Stable models to compute
output: A list of P-Stable models

program ← `invokeAndCreateInput(`*fileName*`);`
if *simplification is enabled* **then** `simplify(`*program*`);`
inputMS ← `createInputMinisat(`*program*`);`
modelsList ← `calculateModels(`*program, n, inputMS*`);`
`return` *modelsList*;

Algorithm 1: PSI external interface

4.2 dataStructures and fileManagement

The package *dataStructures* contains the definitions of the two data structures used in the tool, *Program* and *Rule*, plus some general purpose methods. The class Rule is just a container for two variables, an integer one and an array of integers, respectively the head and the body of the rule to be represented. It also converts the rule in his CNF form, to be returned in form of an array of integers, as stated by the logical equivalence $A \rightarrow B \equiv \neg A \vee B$. The class Program represents the program in memory and contains two private variable, an ArrayList of Rule and a Hashmap where we save the symbol table as returned from Lparse. Also, it provides many methods that help in the process of simplification. The package *fileManagement* is responsible of all the accesses to the file system and to produce the input for Minisat.

4.3 transformations

All the syntactic transformations we studied in Section 3.1 perform in polynomial time. Following the schema of the *simplify* algorithm as depicted in Eiter [8], we implemented a polynomial simplification routine (*Algorithm* 2) which applies the following list of syntactic transformation: **SUB**, **SUC**, **RED**$^-$, **RED**$^+$, **NAF**, **TAUT**, **EQUIV**.

In our implementation the *ChangeInfo* variable does not operate at level of single rule, instead at each main iteration it activates the test of a subset of all implemented rules depending on the changes occurred in the last step. Such changes include the remotion of literals, the introduction of new facts and the reduction of the set of all the heads. Some transformations, as **TAUT** and **EQUIV**, do not need to be tested more than once and for that reason are activated only in the first iteration. In the implementation we made use of some data structures created during the parse of the output of Lparse: the list of facts is maintained in a HashSet which guarantees constant time performances for the basic operations (add, remove, contains and size) assuming the hash function disperses the elements properly among the buckets, the list of head is maintained in a HashMap$< key, value >$ where the key is the integer representing the atom and the value is the number of occurrences of the atom as head (also in this case the basic operations perform in constant time). Under such assumptions plus the one that a literal can be removed in constant time from the body of a clause, the step of testing **SUC**, **RED**$^-$, **RED**$^+$, **NAF**, **TAUT**, **EQUIV** has worst-case running time of the order of $O(n \times m)$, where n is the number of rules in the program and m is the average size of the bodies. The transformation **SUB**, which performs in quadratic time with a brute-force algorithm, has been implemented making use of two sorting steps. The first one sorts all the rules depending on the value of the head as assigned by Lparse, then a second sort

using a stable sorting algorithm is operated based on the size of the bodies. After these orderings, only two rules r, r' such that $H(r) = H(r')$ and $|B(r)| < |B(r')|$ are tested for the applicability of **SUB**. We did not consider the case when $|B(r)| = |B(r')|$ as the applicability of **SUB** implies in this case $r = r'$. This procedure, which has the same worst case complexity of the brute force algorithm, performed quite good in all the tests we did, especially when the number of clauses was massive.

4.4 pstable

This is the fundamental package that actually does all the work of computing minimal and P-Stable models. It implements the algorithm which computes the required P-Stable models of the given program, reading from an input file returned from the class ManageOutput, and repeatedly calling the SAT-solver Minisat. Before giving the algorithm, we repeat the definition of the operator *Neg* [4] and we introduce a new one, *Not*.

Definition 4.1 (Neg)**.** Given a normal logic program P and a set of atoms M, we define $Neg(M)$ as the following set of propositional formulas:

- $\bigwedge \neg\, a$, for all $a \in \mathcal{L}_P \setminus M$.
- $\bigvee \neg\, b$, for all $b \in M$.

```
input  : A normal logic program P.
output: A normal logic program P' such that P ≡_e P'.
var P' ← P, I : changeinfo;
while I ≠ ∅ do
    P' ← ApplyFastRules(P', I);
    P' ← doSUB(P', I);
end
return P';
```

Algorithm 2: Simplification algorithm

Definition 4.2 (Not)**.** Given a set of atoms M, we define $Not(M)$ as the following disjunction of literals:

- $\bigvee \neg\, b$, for all $b \in M$.

Note that in the Definition 4.2 the term literal is used in the context of the propositional calculus. The algorithm which computes all the minimal model is the *Algorithm* 3. As soon as we get a minimal model from the above algorithm we are able to check if it is also a P-Stable one. A new instance of the class Red is created and a new reduced program is obtained adding the disjunction of all the positive atoms of the model but negated. It follows directly from the Definition 2.9 that if the reduced model is unsatisfiable then the model used in the reduction is a P-Stable one. Note that this procedure is a natural consequence of Theorem 2.11. The *Algorithm* 3 executes in exponential time, as we do not make any assumptions on which kind of model is returned from Minisat, the modification of such procedure will be prominent in future works, especially exploiting the special syntactic structure conformation of particular subclasses of the class of normal logic program and the incremental SAT solving technique supported by Minisat.

5 Evaluation

We summarize in this Section the timings relative to the performances of the prototype with respect to some examples taken from the home pages of DLV and Lparse. We must point out that at this stage a comparison with the couple Lparse-Smodels or the DLV system is totally out of place. The machine used for the tests is equipped with a Intel(R) Core(TM)2 Duo CPU T7300 @ 2.00GHz, 2GB of main memory, running the Debian operating system. We selected mainly those examples that were not including the use of constraints as we did not formalize in this paper such concept.

Remark 5.1. We remark by the way that constraints can be encoded including in the program the following clauses [21]

$$c \leftarrow a_1, \ldots, a_m, \textbf{not}\, a_{m+1}, \ldots, \textbf{not}\, a_{m+n}.$$
$$x \leftarrow \textbf{not}\, y, c.$$
$$y \leftarrow \textbf{not}\, z, c.$$
$$z \leftarrow \textbf{not}\, x, c.$$

where c, x, y, z are fresh variables and then substituting every clause

$$\leftarrow a_0, \ldots a_m,\ not\ a_{m+1}, \ldots,\ not\ a_{m+n}.$$

with a new clause

$$c \leftarrow a_0, \ldots a_m,\ not\ a_{m+1}, \ldots,\ not\ a_{m+n}.$$

Such encoding leads to an explosion of the number of clauses in the grounded program making impossible in most of the cases to obtain a model in a reasonable time.

The first six examples are encodings of classical problems available in literature, the 3-colorability of a ladder graph (with 10 nodes and 13 edges in our case), the ancestor problem, the computation of the first 100 Fibonacci's numbers, an encoding of the problem "Who left the zebra out" [1], the problem (with a solution) of finding an Hamiltonian cycle on a graph of 5 nodes and 7 edges and the same problem with no solution on a graph of 7 nodes and 8 edges. The last example is taken from the test cases of the paper "*DES: a challenge problem for nonmonotonic reasoning systems*"

```
input  : A normal logic program p in conjunctive normal form
output: A list of all minimal models of p

M ← minisat(p);
while M is Sat do
    if M is empty then
        add(list, M);
        return list;
    end
    Pc ← P ∪ Neg(M);
    Mc ← minisat(Pc);
    while Mc is Sat do
        Pc ← Pc ∪ Neg(Mc);
        M ← Mc;
        Mc ← minisat(Pc);
    end
    add(list, M);
    p ← p ∪ Not(M);
    M ← minisat(p);
end
return list;
```

Algorithm 3: Minimal models finder

and represent an encoding of the DATA ENCRYPTION STANDARD (DES) encryption function using logic programs [20]. It is are available at `http://www.tcs.hut.fi/Software/smodels/tests/des.html`. All the examples of this section can be downloaded at the prototype home page. The tests are summarized in Table 2, where the headings must be read as: R rules number, L different literals number, Lp Lparse output time, Rr rule removed by transformations, Lr total number of literals removed by transformations, Tt transformations execution time, *TU-time* user time before first model with transformation routine, *NTU-time* user time before first model without transformation routine. The entries of table containing

the symbol > stand for a test that has been stopped during the execution. The encoding of the zebra problem is realized not making use of constraints. It performs very well also behind our expectations, as we tried many encodings of the problem but in the most of the cases we obtained poor results. Even if it is clear that, see for example the timings of D-enc and of the second instance of the Hamiltonian circuit (HamC2), the simplification process helps in some cases, there are also many examples in which the routine does not provide any improvement. The real advantage given from the simplification routine can be only appreciated in presence of computationally onerous examples that still are not handled by the prototype in reasonable time. In future implementations great care must be taken on choosing a correct schema which could also give to the user the possibility to select between the transformations rules to apply. We point out as a informal observation that in some way the simplification process has influence on the order of the models returned by Minisat. A great improvement of the overall process would be the possibility to include Minisat as library and not as an external program. More tests must be done once the constraints satisfaction will be fully included in the code and formally justified.

6 Final considerations

We exposed almost all the transformation rules we could find in literature under P-Stable semantics. As future work we will deal with some syntactic subsets of the set of all normal programs in the context of finding quick procedure to compute P-Stable models. An important step in this direction would be also a correct formalization of the cases when Stable=P-Stable. The implementation schema presented represents at this stage just a naive implementation of P-Stable semantics. We encoded also constraint satisfaction but as we are missing the appropriate formal background we still do not mention it in the official implementation. The performances of the prototype are not bad compared to other more powerful tools, as for example DLV or smodels, if the set of minimal models is not too big.

	R	L	Lp	Rr	Lr	Tt	TU-time	NTU-time
3-col	111	58	0m0.01s	0	173	0m0.00s	0m0.18s	0m0.20s
anc	727	602	0m0.05s	0	1735	0m0.03s	0m0.53s	0m0.36s
Fib	485202	10098	2m5.00s	0	72	2m49.00s	4m48.88s	2m10.18s
Zeb	1283	297	0m0.12s	0	3450	0m1.17s	0m1.82s	0m1.02s
HamC1	282	86	0m0.02s	13	345	0m0.09s	0m1.52s	0m2.69s
HamC2	686	133	0m0.09s	504	14	0m0.34s	8m38.82s	> 20m
D-enc	3840	1842	0m0.42s	637	4081	0m1.00s	0m3.37s	> 30m

Table 2: Tests summary

When such a number increases then the prototype suffers a heavy degradation of performances. We conclude that the procedure followed to look for P-Stable models must be modified at least by using the incremental SAT solving technique with a switch of programming language which could give the opportunity to use Minisat as a library. It is our opinion that better techniques can be obtained from the literature about Stable semantics and SAT-solver implementation. Moreover we need to study more intensively a definition of efficient algorithms for the application of the transformation rules . Stable semantic can be easily encoded, once we have the set of P-Stable models of a given program, using for example the fixed point iteration method. In all the tests made with the prototype we noted that all the models given as output from Minisat were already minimal, if evidences could be given proving that this a standard behavior of the SAT-solver the *Algorithm* 3 could be simplified avoiding the nested while cycle, reducing also the file system accesses.

References

[1] Chitta Baral: *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press, 2003.

[2] Mauricio Osorio Galindo, Juan Antonio Navarro Pérez, José Arrazola Ramírez, Verónica Borja Macías: Logics with Common Weak Completions, *Journal of Logic and Computation*, 2006. doi: 10.1093/logcom/exl013.

[3] Mordechai Ben-Ari: *Mathematical Logic for Computer Science*, Springer, 2005.

[4] Alejandra López Fernández: Implementing Pstable, *Workshop in Logic, Language and Computation*, 2006

[5] Tommi Syrjänen: *Lparse 1.0 User Manual*, web address `http://www.tcs.hut.fi/Software/smodels/`.

[6] DIMACS: *Satisfiability Suggested Format*, available at the address `http://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/satformat.ps`.

[7] Mauricio Osorio Galindo, Juan Antonio Navarro Pérez, José Arrazola Ramírez: Applications of Intuitionistic Logic in Answer Set Programming, *Theory and Practice of Logic Programming* (2004), 4: 325-354 Cambridge University Press. doi: 10.1017/S1471068403001881

[8] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. Simplifying Logic Programs under Uniform and Strong Equivalence. In: *Proc. 7th International Conference on Logic Programming and Non-monotonic Reasoning* (LPNMR-7), I. Niemela and V. Lifschitz, (eds), LNCS, 2004 Springer.

[9] Mauricio Osorio, Juan Antonio Navarro, José Arrazola. Equivalence in Answer Set Programming. In *Proc. LOPSTR 2001*, LNCS 2372, pp. 5775. Springer, 2001.

[10] K. Wang and L. Zhou. Comparisons and Computation of Well-founded Semantics for Disjunctive Logic Programs. *ACM Transactions on Computational Logic*. To appear. Available at: `http://arxiv.org/abs/cs.AI/0301010`, 2003.

[11] José Luis Carballido, Mauricio Osorio, José Ramón Arrazola: Equivalence for the G'_3-stable models semantics, *Proceedings of the 3rd Latin-American Workshop on Non-Monotonic Reasoning* (LANMR-2007), Puebla, Mexico, September 17-19, 2007.

[12] Mauricio Osorio Galindo: GLukG logic and its application for Non-Monotonic Reasoning, *Proceedings of the 3rd Latin-American Workshop on Non-Monotonic Reasoning* (LANMR-2007), Puebla, Mexico, September 17-19, 2007.

[13] M. Osorio, J. A. Navarro, J. Arrazola, V. Borja: Ground non-monotonic modal logic S5: New Result. *Journal of Logic and Computation*, 15(5):787-813, 2005.

[14] M. Osorio, J. A. N. Pérez, J. R. A. Ramírez, and V. B. Macías: Logics where logical weak completions agree, *Journal of Logic and Computation*, December 2006; 16: 867 - 890.

[15] M. Osorio, J. L. Carballido: Brief study of $G_3^{'}$ logic, *Journal of Applied Non-Classical Logics*. Volume 18 No. 4/2008, page 79 to 103.

[16] J. L. Carballido: *Phd Dissertation at BUAP*, 2008, in progress.

[17] Juan Carlos Nieves, Mauricio Osorio: Inferring preferred extensions by Pstable semantics, *Proceedings of the 3rd Latin-American Workshop on Non-Monotonic Reasoning* (LANMR-2007), Puebla, Mexico, September 17-19, 2007.

[18] M. Gelfond and V. Lifschitz: The stable model semantics for logic programming, *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, 1988, pp. 1070-1080.

[19] J. Dix, M. Osorio, C. Zepeda: A general theory of confluent rewriting systems for logic programming and its applications. *Ann. Pure Appl. Logic* 108(1-3): 153-188 (2001).

[20] M. Hietalahti,F. Massacci and I. Niemel: DES: a challenge problem for nonmonotonic reasoning systems. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, Breckenridge, Colorado, USA.

[21] M. Osorio, A. Fernández: Expressing the Stable semantics in terms of the Pstable semantics, *Workshop in Logic, Language and Computation*, Apizaco, Tlaxcala, Mxico, 13th - 14th November 2006.