A Schema for Generating Update Semantics

José Luis Carballido Carranza

Benemérita Universidad Autónoma de Puebla
jlcarballido7@gmail.com

Claudia Zepeda, Guillermo Flores

Benemérita Universidad Autónoma de Puebla
czepedac@gmail.com

***Abstract*—In this paper, we present a general schema for defining new update semantics. This schema takes as input any basic logic programming semantics, such as the stable semantics, the p-stable semantics or the MM^r semantics, and gives as output a new update semantics. The schema proposed is based on a concept called minimal generalized S models, where S is any of the logic programming semantics. Each update semantics is associated to an update operator. We also present some properties of these update operators.**

Keywords Update semantics, Logic Programming semantics, update properties.

I. Introduction

Updating, by definition means that there is new information that must be added to the older, and some information could be changed. Intelligent agents use this, in order to bring new knowledge to their knowledge base. But there is a main problem that updates can present, and it is inconsistency. So, it is important to use an approach to avoid inconsistencies in the knowledge base. For instance, it could be that in an initial moment we can infer a from a knowledge base (KB), and later the KB is updated with the new information $-a$ (where $-$ denotes negation). It is easy to see, that if we only take the union of the initial KB and $-a$, we will have an inconsistency. Then it is useful to apply an update approach that avoids the inconsistency and now allows to infer $-a$ since the newer knowledge has priority over the older. Currently there are several approaches in non-monotonic reasoning dealing with updates, such as [10, 16, 5].

As part of the contribution of this paper, we propose an schema for generating update semantics. This schema takes as input any basic logic programming semantics, such as the *stable semantics* [11], the *p-stable semantics* [17] or the MM^r semantics [13], and gives as output a new update semantics.

It is natural to consider the stable semantics since many approaches to updating have been based on it, see for example [10, 16, 5].

On the other hand, the p-stable semantics is another option to study updating. It has the advantage of providing models that coincide with classical models in many cases. We can make this clear with the following example. Let $P_1 = \{a \leftarrow \neg b, a \leftarrow b\}$ and $P_2 = \{b \leftarrow a\}$. From a classical logic point of view and considering that $\neg$ denotes classical negation, we would expect that $\{a, b\}$ corresponds to the result of updating P_1 with P_2. However, when we apply the approach in [10] based on stable semantics to update P_1 with P_2 there is no model; whereas our schema proposed with the p-stable semantics gives an update semantics that returns $\{a, b\}$ as an update model for the example.

Besides, in [14] it is shown that the p-stable semantics of normal programs can express any problem that can be expressed in terms of the stable semantics of disjunctive programs.

We also consider the MM^r semantics for several reasons. First, any normal program always has MM^r models. Second, it agrees with the Revised Stable models semantics defined by Pereira and Pinto for all the examples they present in their work [18], suggesting that both semantics may coincide for normal logic programs. The coincidence is important since the Revised Stable model semantics has the property of being a relevant semantics. One of the main implications of relevance is that it allows us to define top-down algorithms for answering queries from a knowledge base, this means that relevance allows us to split the original program into subprograms such that finding a model to answer a query can be reduced to finding the models of subprograms [18, 6, 7]. Third the MM^r has been used in the context of argumentation semantics, since it can identify the attack-dependencies that exist in an argumentation framework [13, 3].

Currently there exists a software implementation of the p-stable semantics and the MM^r semantics at `http://aplicacionesia.cs.buap.mx/∼arkerz/` (Windows version) and at
`http://sites.google.com/site/computingpstablesemantics/downloads` (Linux version).

The schema proposed is based on a concept called *minimal generalized S models*, where S is any of the mentioned logic programming semantics. The definition of minimal generalized S models is inspired by a concept called minimal generalized answer sets of abductive programs [12]. The semantics of minimal generalized answer sets is based on the stable semantics. The minimal generalized answer sets have been used to restore consistency [12, 2], to obtain the preferred plans of planning problems [21], to get the preferred extensions of an argument framework [21], and to define update operators [20]. Hence, we consider that minimal generalized S models can also have similar applications and be an alternative to those applications that use minimal generalized answer sets.

Each update semantics is associated to an update operator. Here we also present some properties of these update operators. These properties correspond to the properties of the update operator defined and analized by Eiter et al. [9] and J. J. Alferes et al. in [1], except for one of them called *independent parts property.* This last property refers to the general principle that asserts that completely independent parts of a program should not interfere with each other.

In section we summarize some basic concepts and definitions used to understand this paper. In section we review the minimal generalized S models. In section we present our schema for defining new update semantics and some formal properties. Finally, in section we present some conclusions.

II. Background

In this section, we define the syntax of the logic programs that we will use in this paper. In terms of logic programming semantics, we present the definition of the stable model semantics, the p-stable model semantics, and the MM^r semantics.

A. Logic programs

We use the language of propositional logic in the usual way. We consider *propositional symbols*: $p, q, \ldots$; *propositional connectives*: $\wedge, \vee, \rightarrow, \neg, -$; and *auxiliary symbols*: '(','(',','. Well formed propositional formulas are defined as usual. We consider two types of negation: strong or classical negation (written as $-$) and negation-as-failure (written as $\neg$). Intuitively, $\neg a$ is true whenever there is no reason to believe a, whereas $-a$ requires a proof of the negated atom. An *atom* is a propositional symbol. A *literal* is either an atom a or the strong negation of an atom $-a$.

A *normal* clause is a clause of the form $a \leftarrow b_1 \wedge \ldots \wedge b_n \wedge \neg b_{n+1} \wedge \ldots \wedge \neg b_{n+m}$ where a and each of the b_i are atoms for $1 \leq i \leq n+m$. In a slight abuse of notation we will denote such a clause by the formula $a \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^-$ where the set $\{b_1, \ldots, b_n\}$ will be denoted by $\mathcal{B}^+$, and the set $\{b_{n+1}, \ldots, b_{n+m}\}$ will be denoted by $\mathcal{B}^-$. Given a normal clause $a \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^-$, denoted by r, we say that $a = H(r)$ is the *head* and $\mathcal{B}^+(r) \cup \neg\mathcal{B}^-(r)$ is the *body* of the clause.

A clause with an empty body is called a *fact*; and a clause with an empty head is called a *constraint.* Facts and constraints are also denoted as $a \leftarrow$ and $\leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^-$ respectively. We define a *normal logic program* P, as a finite set of normal clauses. The signature of a normal logic program P, denoted as $\mathcal{L}_P$, is the set of atoms that occur in P. Given a set of atoms M and a signature $\mathcal{L}$, we define $\neg\widetilde{M} = \{\neg a \mid a \in \mathcal{L} \setminus M\}$. Since we shall restrict our discussion to propositional programs, we take for granted that programs with predicate symbols are only an abbreviation of the ground program. From now on, by *program* we will mean a normal logic program when ambiguity does not arise.

In our programs we will manage the strong negation $-$ as follows: each atom $-a$ is replaced by a new atom symbol a' which does not appear in the language of the program and we add the constraint $\leftarrow a \wedge a'$ to the program.

B. Logic programming semantics

Here, we present the definitions of three logic programming semantics. Note that we only consider 2-valued logic programming semantics.

Definition 1. *A logic programming semantics S is a mapping from the class of all programs into the power set of the set of (2-valued) models.*

We sometimes refer to *logic programming semantics* as *semantics*, when no ambiguity arises. The semantics that we consider in this paper are: the MM^r semantics [13] that is based on the *the minimal model semantics* (denoted by MM), *the stable model semantics* [11] (denoted by *stable*), and *the p-stable model semantics* [17] (denoted by *p-stable*). We will review these semantics in the next subsections. From now on, we assume that the reader is familiar with the notion of an *interpretation* and *validity* [19].

When considering any particular semantics of a normal program with constraints $P \cup R$ (R is the set of constraints), we will understand the models given by that semantics of the program P that make the clauses of R valid in the sense of classical logic.

Stable semantics

The stable semantics was defined in terms of the so called *Gelfond-Lifschitz reduction* [11] and it is usually studied in the context of syntax dependent transformations on programs. The following definition of a stable model for normal programs was presented in [11].

Definition 2. *Let P be any program. For any set $M \subseteq \mathcal{L}_P$, let P^M be the definite program obtained from P by deleting each rule that has a literal $\neg l$ in its body with $l \in M$, and then all literals $\neg l$ in the bodies of the remaining clauses. Clearly P^M does not contain $\neg$, then M is a stable model of P if and only if M is a minimal model of P^M.*

Example 3. *Let $M = \{b\}$ and P be the following program: $\{b \leftarrow \neg a,\ \ c \leftarrow \neg b,\ \ b \leftarrow,\ \ c \leftarrow a\}$. Notice that P^M has three models: $\{b\}$, $\{b, c\}$ and $\{a, b, c\}$. Since the minimal model among these models is $\{b\}$, we can say that M is a stable model of P.*

p-stable semantics

Before defining the p-stable semantics (introduced in [17]), we define some basic concepts. Logical inference in classic logic is denoted by $\vdash$. Given a set of proposition symbols S and a theory (a set of well-formed formulas) Γ, $\Gamma \vdash S$ if and only if $\forall s \in S$, $\Gamma \vdash s$. When we treat a program as a theory, each negative literal $\neg a$ is regarded as the standard negation operator in classical logic. Given a normal program P, if $M \subseteq \mathcal{L}_P$, we write $P \Vdash M$ when: $P \vdash M$ and M is a classical 2-valued model of P.

The p-stable semantics is defined in terms of a single reduction which is defined as follows:

Definition 4. *[17] Let P be a program and M be a set of literals. We define*

$$RED(P, M) = \{a \leftarrow \mathcal{B}^+ \cup \neg(\mathcal{B}^- \cap M) \mid a \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^- \in P\}$$

Example 5. *Let us consider the program $P_1 = \{a \leftarrow \neg b \wedge \neg c,\ \ a \leftarrow b,\ \ b \leftarrow a\}$ and the set of atoms $M_1 = \{a, b\}$. We can see that $RED(P, M)$ is: $\{a \leftarrow \neg b,\ \ a \leftarrow b,\ \ b \leftarrow a\}$.*

Next we present the definition of the p-stable semantics for normal programs.

Definition 6. *[17] Let P be a program and M be a set of atoms. We say that M is a p-stable model of P if $RED(P, M) \Vdash M$. We use p-stable to denote the semantics operator of p-stable models.*

Example 7. *Let us consider again P_1 and M_1 of Example 5. Let us verify whether M_1 is a p-stable model of P_1. First, we can see that M_1 is a model of P_1, i.e., for each clause C of P_1, M_1 evaluates C to true. We also can verify that $RED(P_1, M_1) \vdash M_1$. Then we can conclude that $RED(P_1, M_1) \Vdash M_1$. Hence, M_1 is a p-stable model of P_1.*

The following examples illustrate how to obtain the p-stable models. The first example shows a program with a single p-stable model, which is also a classical model. The second example shows a program which has no stable models and whose p-stable and classical models are the same.

Example 8. *Let $P = \{q \leftarrow \neg q\}$. Let us take $M = \{q\}$ then $RED(P, M) = \{q \leftarrow \neg q\}$. 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 of P.*

Example 9. *Let $P = \{a \leftarrow \neg b,\ \ a \leftarrow b,\ \ b \leftarrow a\}$. We can verify that $M = \{a, b\}$ models the clauses of P in classical logic. We find that $RED(P, M) = P$. Now, from the first and third clause, 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 $RED(P, M) \models M$. Therefore, M is a p-stable model of P.*

It is worth mentioning that there exists also a characterization of the p-stable semantics in terms of the paraconsistent logic G'_3, interested readers can see [14, 15, 17].

Minimal model semantics

An interpretation M is called a (2-valued) *model* of P if and only if for each clause $c \in P$, $M(c) = 1$. We say that M is a *minimal model* of P if and only if there does not exist a model M' of P such that $M' \subset M$, $M' \neq M$ [19]. We will denote by $MM(P)$ the set of all the minimal models of a given logic program P. Usually MM is called *minimal model semantics.*

Example 10. *Let P be the program $\{a \leftarrow \neg b,\ \ \ b \leftarrow \neg a,\ \ \ a \leftarrow \neg c,\ \ \ c \leftarrow \neg a\}$. As we can see, P has five models: $\{a\}$, $\{b, c\}$, $\{a, c\}$, $\{a, b\}$, $\{a, b, c\}$; however, P has just two minimal models: $\{b, c\}$, $\{a\}$. Hence $MM(P) = \{\ \{b, c\}, \{a\}\ \}$.*

The MM^r semantics

A program P induces a notion of *dependency* between atoms from $\mathcal{L}_P$ [13]. We say that *a depends immediately on b*, if and only if, b appears in the body of a clause in P, such that a appears in its head. The two place relation *depends on* is the transitive closure of *depends immediately on* [13]. The set of dependencies of an atom x, denoted by *dependencies-of*(x), corresponds to the set $\{a \mid x \textit{ depends on } a\}$.

Example 11. *[13] Let us consider the following program,*
$P = \{e \leftarrow e,\ \ \ c \leftarrow c,\ \ \ a \leftarrow \neg b \wedge c,\ \ \ b \leftarrow \neg a \wedge \neg e,\ \ d \leftarrow b\}$.
The dependency relations between the atoms of $\mathcal{L}_P$ are as follows: dependencies-of$(a) = \{a, b, c, e\}$; dependencies-of$(b) = \{a, b, c, e\}$; dependencies-of$(c) = \{c\}$; dependencies-of$(d) = \{a, b, c, e\}$; and dependencies-of$(e) = \{e\}$.

We take $<_P$ to denote the strict partial order defined as follows: $x <_P y$, if and only if, y *depends-on* x and x does not *depend-on* y. By considering the relation $<_P$, each atom of $\mathcal{L}_P$ is assigned an order as follows: An atom x *is of order* 0, if x is minimal in $<_P$. An atom x *is of order* $n + 1$, if n is the maximal order of the atoms on which x depends.

We say that a program P is of order n, if n is the maximum order of its atoms. We can also break a program P of order n into the disjointed union of programs P_i with $0 \leq i \leq n$, such that P_i is the set of clauses for which the head is of order i (*w.r.t.* P). The empty program has order 0. We say that $P_0, \ldots, P_n$ are the *components* of P.

Example 12. *By considering the program P in Example 11, we can see that: d is of order 2, a is of order 1, b is of order 1, e is of order 0, and c is of order 0. This means that P is a program of order 2. The following table illustrates how the program P can be broken into the disjointed union of the following relevant modules or components $P_0 = \{e \leftarrow e,\ \ \ c \leftarrow c\}$, $P_1 = \{a \leftarrow \neg b \wedge c,\ \ \ b \leftarrow \neg a \wedge \neg e\}$, $P_2 = \{d \leftarrow b\}$.*

Next we present a reduction that will be used to define the MM^r semantics.

Let P be a program and $A = \langle T; F\rangle$ be a pair of disjoint sets of atoms. The reduction $R(P, A)$ is obtained by 2 steps [13]:

1. Let $R'(P, A)$ be the program obtained in the following steps:

   (a) We replace every atom x that occurs in the bodies of P by 1 if $x \in T$, and we replace every atom x that occurs in the bodies of P by 0 if $x \in F$;

   (b) we replace every occurrence of $\neg 1$ by 0 and $\neg$ 0 by 1;

   (c) every clause with a 0 in its body is removed;

   (d) finally we remove every occurrence of 1 in the body of the clauses.

2. Given the set of transformations $\mathcal{CS} = \{RED^+, RED^-, Success, Failure, Loop\}$, in [8] it is shown that a normal program P can be reduced to another normal program $norm_{CS}(P)$ after applying those transformations a finite number of times. The program $norm_{CS}(P)$ is unique and is called the normal form of program P with respect to the system CS. We will denote $R(P, A) = norm_{CS}(R'(P, A))$. The definitions of the transformations in $\mathcal{CS}$ are:

   (a) If $r \in P$ and $a \in B^-(r)$ $\nexists r' \in P : H(r') = a$, then

   $$\mathbf{RED}^+(P) = (P \backslash \{r\}) \cup \{H(r) \leftarrow B^+(r) \cup \neg(B^-(r) \backslash \{a\})\}$$

   (b) If $r \in P$ and $a \leftarrow \in P$ such that $a \in B^-(r)$, then

   $$\mathbf{RED}^-(P) = P \setminus \{r\}$$

   (c) If $r \in P$ and $a \leftarrow \in P$ such that $a \in B^+(r)$, then

   $$\mathbf{Succes}(P) = (P \backslash \{r\}) \cup \{H(r) \leftarrow (B^+(r) \backslash \{a\}) \cup \neg B^-(r)\}$$

   (d) If $r \in P$ and $a \in B^+(r)$ $\nexists r' \in P : H(r') = a$, then

   $$\mathbf{Failure}(P) = P \setminus \{r\}$$

   (e) Let M be unique minimal model of the positive program

   $$\mathbf{POS}(P) = \{H(r) \leftarrow B^+(r) : r \in P\}$$

   then

   $$\mathbf{LOOP}(P) = \{r : r \in P, B^+(r) \subseteq M\}$$

Example 13. *[13] Let $Q = \{a \leftarrow \neg b \wedge c,\ b \leftarrow \neg a \wedge \neg e,\ d \leftarrow b,\ b \leftarrow e, m \leftarrow n, n \leftarrow m\}$, and let A be the pair of sets of atoms $\langle\{c\};\{e\}\rangle$. Thus, $R'(Q, A) = \{a \leftarrow \neg b,\ b \leftarrow \neg a,\ d \leftarrow b, m \leftarrow n, n \leftarrow m\}$. Hence, $R(Q, A) = \{a \leftarrow \neg b,\ b \leftarrow \neg a,\ d \leftarrow b\}$.*

Now, in order to define the MM^r semantics, we first define the MM^r_c semantics in terms of the Minimal Model semantics, denoted by MM.

Definition 14. *Given $\boldsymbol{A} = \{A_1 \ldots A_n\}$ where the A_i, $1 \leq i \leq n$ are sets, and $\boldsymbol{B} = \{B_1 \ldots B_m\}$ where the B_j, $1 \leq j \leq m$ are sets, we define $\boldsymbol{A} \uplus \boldsymbol{B} = \{A_i \cup B_j \mid A_i \in \boldsymbol{A}$ and $B_j \in \boldsymbol{B}\}$.*

Definition 15. *[13] We define the associated MM^r_c semantics recursively as follows: Given a program P of order 0, $MM^r_c(P) = MM(P)$. For a program P of order $n > 0$ we define*

$$MM^r_c(P) = \bigcup_{M \in MM(P_0)} \{M\} \uplus MM^r_c(R(P \setminus P_0, \langle M; N \rangle))$$

where $N := (\mathcal{L}_{P_0} \cup \{a \in \mathcal{L}_P \mid a \notin Head(P)\}) \setminus M$.

It is important to eliminate tautologies from the programs, since they can introduce non-desirable models. For example, if P is the program $\{a \leftarrow \neg b,\ b \leftarrow a, b\}$, then the minimal models for this program are $\{a\}$ and $\{b\}$; however, after deleting the second rule, which is a tautology, it is clear that the second set, namely $\{b\}$ is not an intended minimal model. Given a normal program P, we define $Taut(P) = \{a \leftarrow \mathcal{B}^+ \cup \neg \mathcal{B}^- \in P \mid \mathcal{B}^+ \cap \mathcal{B}^- \neq \emptyset$ *or* $a \in \mathcal{B}^+\}$.

Definition 16. *[13] Let P be a normal program. We define $MM^r(P) = MM^r_c(P \setminus Taut(P))$.*

The following example illustrates our two previous definitions.

Example 17. *Let us consider the program $E = \{a \leftarrow \neg b,\ b \leftarrow \neg a,\ p \leftarrow \neg b,\ p \leftarrow \neg p\}$. We are going to compute the $MM^r_c(E)$. According to Definition 15, since E is of order 1, then we need to obtain the following:*
1) $MM(E_0)$,
2) $MM^r_c(R(E \setminus E_0, \langle M; N \rangle))$ for each $M \in MM(E_0)$, and
3) $MM^r_c(E) = \bigcup_{M \in MM(E_0)} \{M\} \uplus MM^r_c(R(E \setminus E_0, \langle M; N \rangle))$.

***Obtaining** $MM(E_0)$**:** Let us see that $E_0 = \{a \leftarrow \neg b,\ b \leftarrow \neg a\}$ is the component of order 0 of program E. Thus $MM(E_0) = MM(E_0) = \{\{a\}, \{b\}\}$.*

***Obtaining** $MM^r_c(R(E \setminus E_0, \langle M; N \rangle))$ **for each** $M \in MM(E_0)$**:** There are two cases to consider.*

Let us consider M to be $\{a\}$. Then E' is the program $R(E \setminus E_0, \langle M; N \rangle)$ with $E \setminus E_0 = \{p \leftarrow \neg b,\ \ p \leftarrow \neg p\}$, and $N = \{b\}$. We can see that $E' = \{p \leftarrow,\ \ p \leftarrow \neg p\}$. Now we need to obtain $MM^r_c(E')$ which is the same as $MM(E') = \{\{p\}\}$. Hence, $\{M\} \uplus MM^r_c(E') = \{\{a\}\} \uplus \{\{p\}\} = \{\{a, p\}\}$.

Let us consider M to be $\{b\}$. Let E' be the program $R(E \setminus E_0, \langle M; N \rangle)$ with $E \setminus E_0 = \{p \leftarrow \neg b,\ \ p \leftarrow \neg p\}$, and $N = \{a\}$. We can see that $E' = \{p \leftarrow \neg p\}$. Now we need to obtain $MM^r_c(E')$ which is the same as $MM(E') = \{\{p\}\}$. Hence, $\{M\} \uplus MM^r_c(E') = \{\{b\}\} \uplus \{\{p\}\} = \{\{b, p\}\}$.

***Obtaining** $MM^r_c(E)$**:** It is easy to verify that $MM^r_c(E) = \bigcup_{M \in MM(E_0)} \{M\} \uplus MM^r_c(R(E \setminus E_0, \langle M; N \rangle)) = \{\{a, p\}, \{b, p\}\}$.*

Since E is a program with no tautologies then $MM^r(E) = MM^r_c(E)$.

Some properties of logic programming semantics

Not all normal programs have p-stable models or stable models [17, 11], although they always have MM^r models [13], that is why it is convenient to have the next definition.

Definition 18. *Let S be any of the three semantics. Let P be a program. We say that P is S consistent if P has at least one S model. We say that P is S inconsistent if P does not have S models.*

Now, we present two notions of equivalence for programs.

Definition 19. *[4] Let S be any of the three semantics. Two programs P_1 and P_2 are equivalent, denoted by $P_1 \equiv_S P_2$, if P_1 and P_2 have the same S models. Two programs P_1 and P_2 are strongly equivalent, denoted by $P_1 \equiv_{SE(S)} P_2$, if $(P_1 \cup P) \equiv_S (P_2 \cup P)$ for every program P. We will drop the subindex S that follows the equivalent symbol whenever no ambiguity arises.*

The following lemma[1] indicates that given a program P and an atom x that does not occur in P, we can define a new program P' such that P and P' are equivalent and $\mathcal{L}_{P'} = \mathcal{L}_P \cup \{x\}$. The two programs must have the same clauses except for one of them. One of the clauses in P' corresponds to one of the clauses in P after adding $\neg x$ to its body. This way, P and P' have the same S models since x does not appear as the head of any clause in P'.

Lemma 20. *Let S be any of the three semantics. Let P be a program and x be an atom, $x \notin \mathcal{L}_P$. Let r be any clause $a \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^-$ in P. Then M is a S model of P iff M is a S model of $(P \setminus \{r\}) \cup \{a \leftarrow \mathcal{B}^+ \cup \neg(\mathcal{B}^- \cup \{x\})\}$.*

III. Minimal generalized S models

The definition of our schema for generate update semantics is based on a concept called Minimal Generalized S models, denoted as *MG S models*, where S is any of the three semantics given in the Section , namely *stable*, $p-stable$, or MM^r semantics.

The intuition behind the MG S models is simple. Given a semantics S, a program P and a set of atoms A, the MG S models of P are the S models of $P \cup \Delta$ that are obtained by adding the minimal subset $\Delta \subseteq A$ to P for which $P \cup \Delta$ has S models.[2] For instance, let us consider the program $P = \{-a,\ a \leftarrow \neg b\}$, $A = \{b, c\}$, and the p-stable semantics, hence $\{b, -a\}$ is one of its MG p-stable models where the minimal subset of A added to P is $\{b\}$. We also can see that that P does not have p-stable models.

Next, we present the definition of abductive logic programs and their semantics in terms of the *minimal explicit generalized S models*. Then, we define the MG S models based on the minimal explicit generalized S models. These definitions are similar to the definitions of syntax and semantics of abductive logic programs as presented in the context of the stable semantics in [2].

Definition 21. *Let S be a semantics. An abductive logic program is a pair $\langle P, A\rangle$ where P is a program and A is a set of atoms, called abducibles. $\langle M, \Delta\rangle$ is an explicit generalized S model, denoted as EG S model, of the abductive logic program $\langle P, A\rangle$ iff $\Delta \subseteq A$ and M is an S model of $P \cup \Delta$.*

We give an ordering among EG S models in order to get the minimal of them.

Definition 22. *Let S be a semantics. Let $T = \langle P, A\rangle$ be an abductive logic program. Let $\langle M_1, \Delta_1\rangle$ and $\langle M_2, \Delta_2\rangle$ be two EG S models of T, we define $\langle M_1, \Delta_1\rangle < \langle M_2, \Delta_2\rangle$ if $\Delta_1 \subset \Delta_2$; this order is called inclusion order. $\langle M, \Delta\rangle$ is a Minimal EG S model, denoted as MEG S model, of T iff $\langle M, \Delta\rangle$ is an EG S model of T and it is minimal w.r.t. inclusion order.*

For practical purposes, given a MEG S model, $\langle M, \Delta\rangle$, we are only interested in its first entry, namely M, and we call it a *Minimal Generalized S model*, denoted as *MG S model*, of an abductive logic program.

Example 23. *Let S be the p-stable semantics. Let $\langle P, A\rangle$ be the abductive logic program where the set of abductive atoms is $A = \{x_1,\quad x_2\}$ and $P = \{b \leftarrow \neg x_1,\qquad a \leftarrow b \wedge \neg x_2,\qquad -a\}$. There are three EG p-stable models of $\langle P, A\rangle$ which are: $\langle\{-a, x_1\}, \{x_1\}\rangle$, $\langle\{-a, b, x_2\}, \{x_2\}\rangle$, and $\langle\{-a, x_1, x_2\}, \{x_1, x_2\}\rangle$. We can see that for $\Delta = \emptyset$ there is no EG p-stable models. Therefore, the MEG p-stable models are $\langle\{-a, x_1\}, \{x_1\}\rangle$ and $\langle\{-a, b, x_2\}, \{x_2\}\rangle$, and the MG p-stable models are $\{-a, x_1\}$ and $\{-a, b, x_2\}$.*

The following lemma presents some results about MEG S models that will be useful in a later section to show the properties of our update operator. The proof of this lemma is straightforward.

Lemma 24. *Let $T = \langle P, A\rangle$ be an abductive logic program such that P is S consistent. Then,*

- *M is a S model of P iff M is a MG S-model of T and*
- *if $\langle M, \Delta\rangle$ is a MEG S model of T then $\Delta = \emptyset$.*

IV. Updates semantics and formal properties

In this section, we define the general schema for generate update semantics based on the concept of MG S models, and we study some of its properties. We use $\odot_S$ to represent the update operator with respect to a semantics S. In order to obtain the $\odot_S$-update models of a pair of logic programs (P_1, P_2), called *update pair*, we define an update logic program, denoted as P. The update logic program is obtained

[1]Its proof is straightforward.

[2]By "adding the minimal subset $\Delta \subseteq A$ to P", we mean that Δ is interpreted as a set of facts defined by its elements.

by joining P'_1 to P_2, where P'_1 is the resulting program from transforming P_1 as follows: at the end of each clause of P_1 which is not a constraint we add the negation-as-failure of an abducible (a new atom). The intuition behind the transformation applied to a program P_1 consists in weakening the knowledge in P_1 when giving more relevance to the knowledge contained in P_2 whose clauses are not modified.

Definition 25. *Let (P_1, P_2) be an update pair over $\mathcal{L}_{P_1 \cup P_2}$ such that the number of clauses in P_1 that are not constraints is n. Let $\mathcal{L}^*_{P_1 \cup P_2} = \mathcal{L}_{P_1 \cup P_2} \cup A$ where A is a set of n new abducible atoms, namely $A = \{a_i, 1 \leq i \leq n \mid a_i$ is an atom, $a_i \notin \mathcal{L}_{P_1 \cup P_2}$ and $a_i \neq a_j$ if $i \neq j\}$. We define the update logic program P of (P_1, P_2) over $\mathcal{L}^*_{P_1 \cup P_2}$, as the program consisting of the following clauses:*

1. *all constraints in P_1,*
2. *the clauses $a \leftarrow \mathcal{B}^+ \cup \neg(\mathcal{B}^- \cup \{a_i\})$ if $r_i = a \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^- \in P_1$, $1 \leq i \leq n$ and $a_i \in A$,*
3. *all clauses $r \in P_2$.*

We define the abductive logic program of P as follows: $T = \langle \mathsf{P}, A \rangle$.

In this way, given a semantics S, the intended $\odot_S$-update models of a pair of logic programs (P_1, P_2) are obtained by removing the abducible atoms from the MG S models of the abductive logic program $\langle \mathsf{P}, A \rangle$. Finally, the $\odot_S$-update models are chosen as those that contain more information, i.e. maximal in the sense of inclusion of sets, from the intended $\odot_S$-update models.

Definition 26. *Let S be a semantics. Let (P_1, P_2) be an update pair over $\mathcal{L}_{P_1 \cup P_2}$ and T its abductive logic program. Then, $M \subseteq \mathcal{L}_{P_1 \cup P_2}$ is an intended $\odot_S$-update model of (P_1, P_2) if and only if $M = M' \cap \mathcal{L}_{P_1 \cup P_2}$ for some MG S model M' of T. In case M is an intended $\odot_S$-update model of (P_1, P_2) and is maximal among all intended $\odot_S$-update models of (P_1, P_2) w.r.t. inclusion order, then M is an $\odot_S$-update model of (P_1, P_2).*

We can illustrate our semantics with the following example.

Example 27. *Let S be the p-stable semantics. Let (P_1, P_2) be an update pair over $\{a, b\}$ where, P_1 and P_2 are the following logic programs, $P_1 = \{b \leftarrow, \quad a \leftarrow b\}$ and $P_2 = \{-a \leftarrow\}$. We can see that the update logic program P of (P_1, P_2) over $\mathcal{L}^*_{P_1 \cup P_2}$ corresponds to the program P of Example 23 where the x_i are the abducible a_i. The intended $\odot_{p-stable}$-update models of (P_1, P_2) are $\{-a\}$ and $\{-a, b\}$; and its only $\odot_{p-stable}$-update model is $\{-a, b\}$.*

Now, we show that our update operator ($\odot_S$) satisfies several formal properties. These properties have been deeply analyzed, in the context of stable semantics, by several authors such as J. J. Alferes et al. in [1] or T. Eiter in [10], except for the last one. We will see that all the properties are expressed in terms of equivalence, hence it is useful to recall the two notions of equivalence for logic programs given in Definition 19. Since the S models of a logic program are sets of literals, we can see easily that $\equiv$ represents an equivalence relation, and the logic programs P_1 and P_2 can be of any kind defined in this paper.

The following definition is used to define the last of our properties.

Definition 28. *Let S be a semantics. Let (P_1, P_2) be a pair of logic programs over $\mathcal{L}_{P_1 \cup P_2}$. We define the update semantic function of (P_1, P_2) as follows:*
$SEM_{\odot_S}(P_1, P_2)$[3] *$= \{M \mid M$ is an $\odot_S$-update model of $(P_1, P_2)\}$.*

Now we define the properties for $\odot_S$ when S is the stable or p-stable semantics. In the case of the MM^r semantics these properties have not been verified, the study of them are the topic of future work. Since the intuition behind the first six properties is easy, we only give a deeper explanation about the last property below. For any of the semantics S we have the following properties.

P1. *Initialisation:* If P is a logic program then $\emptyset \odot_S P \equiv P$.

P2. *Strong consistency:* Let P_1 and P_2 be logic programs. Suppose $P_1 \cup P_2$ has at least one p-stable model. Then $P_1 \odot_S P_2 \equiv P_1 \cup P_2$.

P3. *Idempotence:* If P is a logic program then $P \odot_S P \equiv P$.

P4. *Weak noninterference:* If P_1 and P_2 are logic programs defined over disjoint alphabets, and both of them have p-stable models or do not, then $P_1 \odot_S P_2 \equiv P_2 \odot_S P_1$.

P5. *Weak irrelevance of syntax:* Let P, P_1 and P_2 be logic programs under $\mathcal{L}_{\mathsf{P}}$. If $P_1 \equiv_{SE} P_2$ then $P \odot_S P_1 \equiv P \odot_S P_2$.

P6. *Augmented update:* Let P_1 and P_2 be logic programs such that $P_1 \subseteq P_2$. Then $P_1 \odot_S P_2 \equiv P_2$.

P7. *Independent parts property.* Let $J_1 = (P_1, P'_1)$ and $J_2 = (P_2, P'_2)$ such that $(\mathcal{L}_{J_1} \cap \mathcal{L}_{J_2}) = \emptyset$. Then $SEM_{\odot_S}((P_1 \cup P_2), (P'_1 \cup P'_2)) = SEM_{\odot_S}(J_1) \uplus SEM_{\odot_S}(J_2)$.

Property **P7** indicates that our update operator does not violates the general principle that completely independent parts of a logic program should not interfere with each other. Hence the property **P7** of operator $\odot$ indicates that if we update the union of a pair of logic programs $(P_1 \cup P_2)$ by the union of a different pair of logic programs $(P'_1 \cup P'_2)$ such that P_1 and P'_1 are defined under a different language from the language of logic programs P_2 and P'_2 then, the

[3]Let us notice that $SEM_{\odot_S}(P_1, P_2)$ is a set of sets.

result can be also obtained from a particular union of the update of P_1 by P_1' and the update of P_2 by P_2'. This particular union of updates corresponds to our Definition 28.

The next example is taken from [16], where it is used for different purposes.

Example 29.

Let P_1 be:	*Let P_1' be:*
$openSchool \leftarrow .$	$-openSchool \leftarrow holiday.$
$holiday \leftarrow \neg workday.$	$workday \leftarrow \neg holiday.$
Let P_2 be:	*Let P_2' be:*
$seeStars \leftarrow .$	$-seeStars \leftarrow .$

Let S be the p-stable semantics. Let $J_1 = (P_1, P_1')$, $J_2 = (P_2, P_2')$, and $J = ((P_1 \cup P_2), (P_1' \cup P_2'))$. We can see that $(\mathcal{L}_{J_1} \cap \mathcal{L}_{J_2}) = \emptyset$.
According to independent parts property we have that, $SEM_{\odot_{p-stable}}((P_1 \cup P_2), (P_1' \cup P_2')) = SEM_{\odot_{p-stable}}(J_1) \uplus SEM_{\odot_{p-stable}}(J_2)$ *since*
$SEM_{\odot_{p-stable}}((P_1 \cup P_2), (P_1' \cup P_2')) = \{\{openSchool, workday, -seeStars\}\} = \{\{openSchool, workday\}\} \uplus \{\{-seeStars\}\} = SEM_{\odot_{p-stable}}(J_1) \uplus SEM_{\odot_{p-stable}}(J_2)$.

Theorem 30. *The update operator ($\odot_S$) satisfies properties,* **P1**, **P2**, **P3**, **P4**, **P5**, **P6**, *and* **P7** *when S is stable or p-stable semantics.*

Proof. We present the proof for the p-stable semantics. Properties **P1** to **P6** for stable semantics are proved in [20]. The proof of property **P7** for the stable semantics is similar to the one presented here for the p-stable semantics.

First, it is straightforward to verify that given a p-stable consistent program P, if M is p-stable model of P then there is not another p-stable model M' of P such that $M' \subset M$. So, by this last fact and by Lemma 24, it is also straightforward to verify that given an abductive logic program $\langle P, A\rangle$, where P is p-stable consistent, then if M is a MG p-stable model of $\langle P, A\rangle$ then there is not another MG p-stable model M' of $\langle P, A\rangle$ such that $M' \subset M$.
(**P1. Initialisation**): $\emptyset \odot P = P$ by construction. Hence $\emptyset \odot_{p-stable} P \equiv P$.
(**P2. Strong consistency**): Let $Q = (P_1 \cup P_2)$ such that Q is p-stable consistent. Let $J = (P_1, P_2)$. We must prove that M is an $\odot_{p-stable}$ update model of J iff M is a p-stable model of Q.
Let us notice that programs Q and P have the same clauses except for some of them, namely in P there are some clauses that have an abducible atom (a new atom) in their body and these atoms do not occur in Q. So when we apply iteratively Lemma 20, two things are certain:

(1) S is a p-stable model of Q, iff S is also a p-stable model of P, and

(2) if Q is p-stable consistent, then P is p-stable consistent too.

($\Rightarrow$) By hypothesis M is an $\odot_{p-stable}$ update model of J, then by Definition 26, there exists a MG p-stable model M' of the abductive logic program of J, $\langle \mathsf{P}, A\rangle$, such that $M = M' \cap \mathcal{L}_J$. Then by Definition 22, there exists Δ, $\Delta \subseteq A$ such that $\langle M', \Delta\rangle$ is a MG p-stable model of $\langle \mathsf{P}, A\rangle$, where $M = M' \cap \mathcal{L}_J$.

By hypothesis, Q is p-stable consistent then, by (2) P is p-stable consistent too. Hence applying Lemma 24, it is possible to verify that $\Delta = \emptyset$ and $M' = M$. So $\langle M, \emptyset\rangle$ is a MEG p-stable model of $\langle \mathsf{P}, A\rangle$. Finally by Definition 21, M is a p-stable model of P. Thus by (1) we have that M is a p-stable model of Q.
($\Leftarrow$) Let M be a p-stable model of Q. By (1), M is a p-stable model of P. By Lemma 24, M is a MG p-stable model of the abductive logic program of J, $\langle \mathsf{P}, A\rangle$. By Lemma 24, $M \cap A = \emptyset$. Hence by Definition 26, M is an $\odot_{p-stable}$ update model of J.
(**P3. Idempotence**): If P does not have p-stable models, then neither does $P \odot_{p-stable} P$. If P has p-stable models, then $P \cup P$ does, hence by Strong Consistency, $P \cup P \equiv P \odot_{p-stable} P$. Hence in each case $P \odot_{p-stable} P \equiv P$.
(**P4. Weak noninterference**): If each of P_1 and P_2 lacks of p-stable models then the update (in any order) lacks of p-stable models. If P_1 and P_2 have p-stable models, then $P_1 \cup P_2$ does too — because they are defined over disjoint alphabets. By Strong Consistency, $P_1 \cup P_2 \equiv P_1 \odot_{p-stable} P_2$. Also $P_2 \cup P_1 \equiv P_2 \odot_{p-stable} P_1$. Hence, $P_1 \odot_{p-stable} P_2 \equiv P_2 \odot_{p-stable} P_1$.
(**P5. Weak irrelevance of syntax**): Let P, P_1, and P_2 be logic programs under the same language $\mathcal{L}$. Since $P_1 \equiv_{SE} P_2$, then for every program P, $P \cup P_1$ is strongly equivalent to $P \cup P_2$. Thus, $(P \cup A) \cup P_1$ and $(P \cup A) \cup P_2$ have exactly the same p-stable models. Thus, $P \odot_{p-stable} P_1$ and $P \odot_{p-stable} P_2$ have exactly the same EG p-stable models. Therefore, $P \odot_{p-stable} P_1$ and $P \odot_{p-stable} P_2$ have exactly the same MG p-stable models. Hence, $P \odot_{p-stable} P_1 \equiv P \odot_{p-stable} P_2$.
(**P6. Augmented update**): If P_2 does not have p-stable models, neither does $P_1 \odot_{p-stable} P_2$. If P_2 has at least one p-stable model and $P_1 \subseteq P_2$ then, $(P_1 \cup P_2)$ has at least one p-stable model too. By strong consistency $P_1 \odot_{p-stable} P_2 \equiv P_1 \cup P_2$. Hence in each case $P_1 \odot_{p-stable} P_2 \equiv P_2$.
(**P7. Independent parts**): Let $J_1 = (P_1, P_1')$, $J_2 = (P_2, P_2')$ such that $(\mathcal{L}_{J_1} \cap \mathcal{L}_{J_2}) = \emptyset$, and $J = ((P_1 \cup P_2), (P_1' \cup P_2'))$. Let M_1 and M_2 be a $\odot_{p-stable}$ update model of J_1 and a $\odot_{p-stable}$ update model of J_2 respectively. It is clear that M_1 and M_2 are disjoint, since $(\mathcal{L}_{J_1} \cap \mathcal{L}_{J_2}) = \emptyset$. We have to prove that M is a $\odot_{p-stable}$ update model of J iff $M = M_1 \cup M_2$.

($\Rightarrow$) By Definition 26, if M is a $\odot_{p-stable}$ update model of J, then there exists M', a MG p-stable model of $\langle \mathsf{P}, B\rangle$, such that $M = M' \cap \mathcal{L}_J$. Then by Definition 22, there exists Δ, $\Delta \subset B$ such that $\langle M', \Delta\rangle$ is a EG p-stable model of $\langle \mathsf{P}, B\rangle$ and it is minimal. By Definition 21, M' is a p-stable

model of $\mathsf{P} \cup \Delta$.
Moreover, since $(\mathcal{L}_{J_1} \cap \mathcal{L}_{J_2}) = \emptyset$, we can verify the following:

(1) $\mathsf{P} = \mathsf{P}_1 \cup \mathsf{P}_2$[4],

(2)$\Delta = \Delta_1 \cup \Delta_2$ such that $\Delta_1 = \Delta \cap \mathcal{L}_{J_1}$, $\Delta_2 = \Delta \cap \mathcal{L}_{J_2}$ and $\Delta_1 \cap \Delta_2 = \emptyset$,

(3)$M' = M'_1 \cup M'_2$ such that M'_1 is a p-stable model of $\mathsf{P}_1 \cup \Delta_1$ and M'_2 is a p-stable model of $\mathsf{P}_2 \cup \Delta_2$.

Now by Definition 21, $\langle M'_1, \Delta_1 \rangle$ is a MEG p-stable model of $\langle \mathsf{P}_1, B_1 \rangle$ and $\langle M'_2, \Delta_2 \rangle$ is a MGE p-stable model of $\langle \mathsf{P}_2, B_2 \rangle$ where B_1 is the set of abducible atoms of P_1 and B_2 is the set of abducible atoms of P_2.
Finally by Definition 26, $M_1 = M'_1 \cap \mathcal{L}_{J_1}$ and $M_2 = M'_2 \cap \mathcal{L}_{J_2}$ are $\odot_{p-stable}$ update model of J_1 and J_2 respectively.
$(\Leftarrow)$ This proof is similar to the proof of the first part above. Taking into account that $\mathsf{P} = \mathsf{P}_1 \cup \mathsf{P}_2$; and if $\Delta = \Delta_1 \cup \Delta_2$ then there exists a p-stable model $M' = M'_1 \cup M'_2$ of $\mathsf{P} \cup \Delta$ such that M'_1 as a p-stable model of $\mathsf{P}_1 \cup \Delta_1$ and M'_2 as a p-stable model of $\mathsf{P}_2 \cup \Delta_2$. $\square$

V. Conclusions

Our general schema for defining new update semantics takes as input any basic logic programming semantics S and gives as output a new update semantics. The schema uses minimal generalized S models, where S is any of the logic programming semantics. Each update semantics is associated to an update operator. We also presented properties for the update operator which are valid for the stable and p-stable semantics. The study of those properties for the MM^r semantics as well as the extension of our results to other semantics along with a comparative study of them are topics to be developed in future work.

Acknowledgement: This research has been supported by the Fondo Sectorial SEP-CONACyT, Ciencia Básica Project (Register 101581).

References

[1] J. J. Alferes, F. Banti, A. Brogi, and J. A. Leite. The refined extension principle for semantics of dynamic logic programming. Studia Logica, 79(1):7–32, 2005.

[2] M. Balduccini and M. Gelfond. Logic Programs with Consistency-Restoring Rules. In P. Doherty, J. McCarthy, and M.-A. Williams, editors, International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series, Mar 2003.

[3] P. Baroni, M. Giacomin, and G. Guida. SCC-recursiveness: a general schema for argumentation semantics. Artificial Intelligence, 168:162–210, October 2005.

[4] J. L. Carballido, M. Osorio, and J. Arrazola. Equivalence for the G'$_3$-stable models semantics. J. Applied Logic, 8(1):82–96, 2010.

[5] J. Delgrande, T. Schaub, and H. Tompits. A preference-based framework for updating logic programs. pages 71–83.

[6] J. Dix. A classification theory of semantics of normal logic programs: II. weak properties. Fundam. Inform., 22(3):257–288, 1995.

[7] J. Dix and M. Müller. Partial evaluation and relevance for approximations of stable semantics. In ISMIS, volume 869 of Lecture Notes in Computer Science, pages 511–520. Springer, 1994.

[8] J. Dix, M. Osorio, and 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.

[9] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. Considerations on updates of logic programs. In JELIA '00: Proceedings of the European Workshop on Logics in Artificial Intelligence, pages 2–20, London, UK, 2000. Springer-Verlag.

[10] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On properties of update sequences based on causal rejection. Theory and Practice of Logic Programming, 2(6):711–767, 2002.

[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] A. C. Kakas and P. Mancarella. Generalized stable models: a semantics for abduction. In Proceedings of ECAI-90, pages 385–391. IOS Press, 1990.

[13] J. C. Nieves, M. Osorio, and C. Zepeda. A schema for generating relevant logic programming semantics and its applications in argumentation theory. Accepted in Fundamenta Informatica, 106(2-4):295–319, 2011.

[14] 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.

[15] M. Osorio and J. L. Carballido. Brief study of G'$_3$ logic. Journal of Applied Non-Classical Logic, 18(4):79–103, 2009.

[16] M. Osorio and V. Cuevas. Updates in answer set programming: An approach based on basic structural properties. Theory and Practice of Logic Programming, 7(04):451–479, July 2007.

[17] 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.

[18] L. M. Pereira and A. M. Pinto. Revised stable models - a semantics for logic programs. In C. Bento, A. Cardoso, and G. Dias, editors, EPIA, volume 3808 of Lecture Notes in Computer Science, pages 29–42. Springer, 2005.

[19] D. van Dalen. Logic and structure. Springer-Verlag, Berlin, 3rd., aumented edition edition, 1994.

[20] F. Zacarias, M. O. Galindo, J. C. A. Guadarrama, and J. Dix. Updates in Answer Set Programming based on structural properties. In Proceedings of the 7th International Symposium on Logical Formalizations of Commonsense Reasoning. Dresden University Technical Report, pages 213–219, Corfu, Greece, May 2005. TU-Dresden, Fakultt Informatik.

[21] C. Zepeda, M. Osorio, J. C. Nieves, C. Solnon, and D. Sol. Applications of preferences using answer set programming. In Answer Set Programming: Advances in Theory and Implementation (ASP 2005), pages 318–332, University of Bath, UK, July 2005.

[4]This is possible if we select the appropriate abducibles from P to define P_1 and P_2 (see Definition 25).