Technical Report: *Universitat Politècnica de Catalunya, Software Department*
http://www.lsi.upc.edu/dept/techreps/techreps.html

Weakening the Stable semantics

JUAN CARLOS NIEVES
Universitat Politècnica de Catalunya
Software Department
c/Jordi Girona 1-3, E08034, Barcelona, Spain
(*e-mail:* jcnieves@lsi.upc.edu)

MAURICIO OSORIO
Universidad de las Américas - Puebla
CENTIA
Sta. Catarina Mártir, Cholula, Puebla, 72820 México
(*e-mail:* josorio@mail.udlap.mx)

9 December 2005

Abstract

We report our research on semantics for normal/disjunctive programs. One of the most well known semantics for logic programming is the stable semantics (STABLE). However, it is well known that very often STABLE has no models. In this paper we study the stable semantics and present some new results about it. Furthermore, we introduce a new semantics (that we call D3-WFS-DCOMP) and compare it with STABLE. For normal programs, this semantics is based on a *suitable* integration of WFS and the Clark's Completion. D3-WFS-DCOM has the following appealing properties: First, it agrees with STABLE in the sense that it never defines a non minimal model or a non minimal supported model. Second, for normal programs it extends WFS. Third, every stable model of a disjunctive program P is a D3-WFS-DCOM model of P. Fourth, it is constructed using transformations accepted by STABLE. We also introduce a second semantics that we call D2-WFS-DCOMP. We show that D2-WFS-DCOMP is equivalent to D3-WFS-DCOMP for normal programs but this is not the case for disjunctive programs. We also introduce a third new semantics that insists in the use of implicit disjunctions. We briefly sketch how these semantics can be extended to programs including: explicit negation, default negation in the head of a clause, as well as a *lub* operator (which is the generalization of setof over arbitrary complete lattices). We sketch how to model this *lub* operator using standard disjunctive clauses. However, we can not use the STABLE semantics but instead any of our suggested semantics.

We emphasizes that the ultimate goal of our research is to understand better the STABLE semantics and to suggest solutions to the drawbacks of the stable semantics (that becomes undefined very often).

KEYWORDS: Answer Set Programming, Declarative Programming, Non-Monotonic Reasoning.

1 Introduction

One of the most well known semantics for logic programming is the stable semantics (STABLE) (Gelfond and Lifschitz 1988). However, it is well known that very often

STABLE is too strong, namely that there are programs that do not possess stable models (You and Yuan 1994; Dix et al. 2001; Dix 1995). Let us consider the following example.

Example 1
Let P be a disjunctive logic program:

$\mathtt{d} \vee \mathtt{e} \leftarrow \neg \mathtt{a}.$
$\mathtt{c} \leftarrow \mathtt{c}.$
$\mathtt{b} \leftarrow \mathtt{a}.$
$\mathtt{a} \leftarrow \mathtt{b}.$
$\mathtt{a} \leftarrow \neg \mathtt{b}, \neg \mathtt{c}.$

In this program there are not stable models, however if we use our D3-WFS-DCOMP semantics there is a model of the program P, i.e. {a,b}. We can argue that this intended model makes sense as follows: Since a tautology such as $\mathtt{c} \leftarrow \mathtt{c}$ should not matter[1], we can delete it getting a semantically equivalent program that we call P_1. Then, since c does not appear in the head of any clause of P_1, we can derive $\neg c$ and so we can delete $\neg c$ in the last clause to get a semantically equivalent program P_3[2]. Then, since we have that:

$\mathtt{a} \leftarrow \mathtt{b}.$
$\mathtt{a} \leftarrow \neg \mathtt{b}.$

are clauses in P_3 and by reasoning by cases we derive a.[3] Then, we derive b by simple modus ponens. Finally, d and e can be considered false since a is true.

It has been even argued that STABLE does not define the intended models of a program(Przymusinski 1991). It is also possible that there is not a best semantics (Dung 1992):

> "The fact each approach to semantics of negation has its own strength and weakness suggests that there is probably not a *best* semantics for logic programs".

Nevertheless, STABLE satisfies interesting logic properties as it is shown in (Pearce 1999). In this paper we study the stable semantics and exhibit some properties that it satisfies. Also we point out some problems with this semantics. With this base, we made some variations to the semantics proposed in (Arrazola et al. 1999; Osorio and Zacarias 2000). The obtained semantics, that we call *D3 - WFS - DCOMP* satisfies the following main properties: First, it agrees with STABLE in the sense that it never defines a non minimal model or a non minimal supported model. Second, for normal programs it extends WFS. Third, every stable model of a disjunctive program P is a D3-WFS-DCOM model of P. Fourth, it is constructed using transformations accepted by STABLE.

We also introduce two more semantics and prove some properties of them. We show that our three proposed are equivalent for normal programs. In addition, our

[1] In our point of view, what makes the given tautology to be specially harmless is that c does not occurs in the head of any other clause
[2] Note, that what we have done so far in this example is accepted by the STABLE semantics
[3] Reasoning by cases is not accepted in STABLE.

empirical research suggests that they are suitable for modeling the *setof* operator of PROLOG. We have done an extensive research on this topic in (Osorio and Jayaraman 1999; Osorio et al. 1999; Osorio and Zacarias 2000; Nieves and Jayaraman 2002).

Our paper is structured as follows: In §2, we review the basic foundations. In §3 we prove some properties of stable model and define our new semantics. In §4 we show with modeling the *setof* operator of PROLOG with our semantics using negation as failure also we present how to reduce programs with explicit negation to programs without it. Finally, in §5 we present our conclusions.

2 Background

A signature $\mathcal{L}$ in a finite set of elements that we call atoms. A literal is an atom or the negation of an atom a that we denote as $\neg a$. Given a set of atoms $\{a_1, ..., a_n\}$, we write $\neg\{a_1, ..., a_n\}$ to denote the set $\{\neg a_1, ..., \neg a_n\}$.

A theory is built up using the logical constants $\wedge, \vee, \rightarrow$, and $\neg$. We may denote a (general) clause C as: $a_1 \vee \ldots \vee a_m \leftarrow l_1, \ldots, l_n$, [4] where $m > 0$, $n \geq 0$, each a_i is a propositional atom, and each l_i is a propositional literal. When $n = 0$ the clause is considered as $a_1 \vee \ldots \vee a_m \leftarrow true$[5], where $true$ is a constant atom with its intended interpretation. Sometimes, is better to denote a clause C by $\mathcal{A} \leftarrow \mathcal{B}^+,\ \neg\mathcal{B}^-$, where $\mathcal{A}$ contains all the head atoms, $\mathcal{B}^+$ contains all the positive body atoms and $\mathcal{B}^-$ contains all the negative body atoms. We also use $body(C)$ to denote $\mathcal{B}^+ \cup \neg\mathcal{B}^-$. When $\mathcal{A}$ is a singleton set, the clause reduces to a normal clause. A definite clause ((Lloyd 1987)) is a normal clause lacking of negative literals, that is $\mathcal{B}^- = \emptyset$. A *pure* disjunction is a disjunction consisting solely of positive or solely of negative literals. A (general) program is a finite set of clauses. As in normal programs, we use $HEAD(P)$ to denote the set of atoms occurring in the heads of P. Given a signature $\mathcal{L}$, we write $Prog_{\mathcal{L}}$ to denote the set of all programs defined over $\mathcal{L}$. We use $\models$ to denote the consequence relation for classical first-order logic. We will also consider interpretations and models as usual in classical logic.

Given two theories T_1, T_2 we denote $T_1 \equiv_I T_2$ to say that these theories are intuitionistically equivalent, i.e. $\forall \alpha \in T_2$, then T_1 proves (using intuitionistic logic) α and conversely.

It will be useful to map a program to a normal program. Given a clause $C := \mathcal{A} \leftarrow \mathcal{B}^+, \neg\mathcal{B}^-$, we write *dis-nor(C)* to denote the set of normal clauses:

$\{a \leftarrow \mathcal{B}^+, \neg(\mathcal{B}^- \cup (\mathcal{A} \setminus \{a\}) | a \in \mathcal{A}\}$.

We extend this definition to programs as follows. If P is a program, let *dis-nor(P)* denote the normal program: $\bigcup_{C \in P} dis - nor(C)$. Given a normal program P, we write $Definite(P)$ to denote the definite program that we obtain from P just by removing every negative literal in P. Given a definite program, by $MM(P)$ we mean the unique minimal model of P (that always exists for definite programs, see (Lloyd 1987)).

[4] $l_1, \ldots, l_n$ represents the formula $l_1 \wedge \ldots \wedge l_n$.
[5] Or the simple formula $a_1 \vee \ldots \vee a_m$

We assume to work with disjunctive programs, unless stated otherwise.
We use an example to illustrate the above definitions. Let P be the program:

$\mathtt{p} \vee \mathtt{q} \leftarrow \neg \mathtt{r}$.

$\mathtt{p} \leftarrow \mathtt{s}, \neg \mathtt{t}$.

Then $HEAD(P) = \{\mathtt{p}, \mathtt{q}\}$, and $dis - nor(P)$ consists of the clauses:

$\mathtt{p} \leftarrow \neg \mathtt{r}, \neg \mathtt{q}$.

$\mathtt{q} \leftarrow \neg \mathtt{r}, \neg \mathtt{p}$.

$\mathtt{p} \leftarrow \mathtt{s}, \neg \mathtt{t}$

$Definite(dis - nor(P))$ consists on the clauses:

$\mathtt{p} \leftarrow true$.

$\mathtt{q} \leftarrow true$.

$\mathtt{p} \leftarrow \mathtt{s}$.

MM(Definite(dis-nor(P)))=$\{\mathtt{p}, \mathtt{q}\}$.

What are the minimal requirements we want to impose on a semantics for disjunctive programs? Certainly, we want that disjunctive facts, i.e. clauses in the program with empty bodies to be true. Dually, if an atom does not occur in any head, then its negation should be true. These ideas are straightforward generalizations of the case on normal programs. Following the ideas of (Brass and Dix 1997), we propose the following definition:

Definition 1 (*Semantics*)

- A scenario semantics over a given signature $\mathcal{L}$, is a function such that for every program P over $\mathcal{L}$ it associates a set of models.
- The sceptical semantics induced by a scenario semantics is the set of pure disjunctions true in every model of the scenario semantics[6]. For any program P we define:
  $\text{SEM}_{min}(P) := \{a|\ a \leftarrow\ true \in P\} \cup \{\neg a | a \in \mathcal{L}, a \notin HEAD(P)\}$

Given two theories P_1, P_2, we denote $P_1 \equiv_{SEM} P_2$ whenever the set of models of P_1 w.r.t. the scenario semantics SEM is equal to the set of models of P_2 w.r.t. the scenario semantics SEM.

Definition 2 (*Supported model,(Brass and Dix 1997)*)
A two-valued model I of a (disjunctive) logic program P is supported iff for every ground atom A with I $\models$ A there is a rule $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \neg\mathcal{B}^-$ in P with A $\in \mathcal{A}$, I $\models \mathcal{B}^+ \wedge \neg\mathcal{B}^-$, and I $\not\models \mathcal{A} \setminus \{A\}$.

Definition 3 (*STABLE, (Gelfond and Lifschitz 1988)*)
The Gelfond-Lifschitz transformation (GL-transformation) of a logic program P w.r.t. an interpretation M is obtained from P by deleting

(i) each rule that has a negative literal $\neg B$ in its body with $B \in M$, and
(ii) all negative literals in the bodies of the remaining rules.

[6] The sceptical semantics derives every literal when its associated scenario semantics has no models

Clearly the program $GL(P, M)$ resulting from applying the GL-transformation is negation-free, so $GL(P, M)$ has at least one model. M is a stable model iff M is a minimal model of $GL(P, M)$.

Recently, Pearce (Pearce 1999) has generalized the stable semantics to any propositional theory based on intuitionistic logic. Pearce showed the following (Pearce 1999):

> "A formula is entailed by a program in the stable model semantics if and only if it belongs to every intuitionistically complete and consistent extension of the program formed by adding only negated atoms".

From this characterization of the stable semantics, the generalization of STABLE to arbitrary theories is immediate. From now on, we will assume this definition whenever we mention the stable semantics of some given theory.

The following transformations are defined in (Brass and Dix 1997) and generalize the corresponding definitions for normal programs.

Definition 4 (*Basic Transformation Rules*)
A transformation rule is a binary relation on $Prog_{\mathcal{L}}$. The following transformation rules are called basic. Let a program $P \in Prog_{\mathcal{L}}$ be given.

RED$^+$: Replace a rule $\mathcal{A} \leftarrow \mathcal{B}^+, \neg\mathcal{B}^-$ by $\mathcal{A} \leftarrow \mathcal{B}^+, \neg(\mathcal{B}^- \cap HEAD(P))$.
RED$^-$: Delete a clause $\mathcal{A} \leftarrow \mathcal{B}^+, \neg\mathcal{B}^-$ if there is a clause $\mathcal{A}' \leftarrow true$ such that $\mathcal{A}' \subseteq \mathcal{B}^-$.
SUB: Delete a clause $\mathcal{A} \leftarrow \mathcal{B}^+, \neg\mathcal{B}^-$ if there is another clause $\mathcal{A}_1 \leftarrow \mathcal{B}_1^+, \neg\mathcal{B}_1^-$ such that $\mathcal{A}_1 \subseteq \mathcal{A}$, $\mathcal{B}_1^+ \subseteq \mathcal{B}^+$, $\mathcal{B}_1^- \subseteq \mathcal{B}^-$.

Example 2 (*Transformation*)
Let $\mathcal{L} = \{a, b, c, d, e\}$ and let P be the program:
$\mathtt{a} \vee \mathtt{b} \leftarrow \mathtt{c}, \neg\mathtt{c}, \neg\mathtt{d}.$
$\mathtt{a} \vee \mathtt{c} \leftarrow \mathtt{b}.$
$\mathtt{c} \vee \mathtt{d} \leftarrow \neg\mathtt{e}.$
$\mathtt{b} \leftarrow \neg\mathtt{c}, \neg\mathtt{d}, \neg\mathtt{e}.$
then $HEAD(P) = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, and $SEM_{min}(P) = \{\neg e\}$.
We can apply **RED$^+$** to get the program P_1:
$\mathtt{a} \vee \mathtt{b} \leftarrow \mathtt{c}, \neg\mathtt{c}, \neg\mathtt{d}.$
$\mathtt{a} \vee \mathtt{c} \leftarrow \mathtt{b}.$
$\mathtt{c} \vee \mathtt{d} \leftarrow true.$
$\mathtt{b} \leftarrow \neg\mathtt{c}, \neg\mathtt{d}, \neg\mathtt{e}.$
If we apply **RED$^+$** again, we get program P_2:
$\mathtt{a} \vee \mathtt{b} \leftarrow \mathtt{c}, \neg\mathtt{c}, \neg\mathtt{d}.$
$\mathtt{a} \vee \mathtt{c} \leftarrow \mathtt{b}.$
$\mathtt{c} \vee \mathtt{d} \leftarrow true.$
$\mathtt{b} \leftarrow \neg\mathtt{c}, \neg\mathtt{d}.$
Now, we can apply **SUB** to get program P_3:
$\mathtt{a} \vee \mathtt{c} \leftarrow \mathtt{b}.$
$\mathtt{c} \vee \mathtt{d} \leftarrow true.$
$\mathtt{b} \leftarrow \neg\mathtt{c}, \neg\mathtt{d}.$

The following transformations are defined in (Brass and Dix 1997).

GPPE: *(Generalized Principle of Partial Evaluation)* Suppose P contains $\mathcal{A} \leftarrow \mathcal{B}^+,\ \neg\mathcal{B}^-$ and we fix an occurrence of an atom $g \in \mathcal{B}^+$. Then we replace $\mathcal{A} \leftarrow \mathcal{B}^+,\ \neg\mathcal{B}^-$ by the n clauses $(i = 1, \ldots, n)$

$$\mathcal{A} \cup (\mathcal{A}_i \backslash \{g\}) \leftarrow (\mathcal{B}^+ \backslash \{g\}) \cup {B_i}^+,\ \neg\mathcal{B}^- \cup \neg{B_i}^-$$

where $\mathcal{A}_i \leftarrow {B_i}^+,\ \neg{B_i}^- \in$ P, (for $i = 1, \ldots, n$) are all clauses with $g \in \mathcal{A}_i$. If no such clauses exist, we simply delete the former clause.

TAUT: *(Tautology)* Suppose P contains a clause of the form: $\mathcal{A} \leftarrow \mathcal{B}^+,\ \neg\mathcal{B}^-$ and $\mathcal{A} \cap \mathcal{B}^+ \neq \emptyset$, then we delete the given clause.

Let $\mathcal{CS}_1$ be the rewriting system which contains, besides the basic transformation rules, the rules GPPE and TAUT. This system was introduced in (Brass and Dix 1997) and is confluent and terminating as shown in (Brass and Dix 1998). We write $res_{\mathcal{CS}_1}(P)$ to denote the residual program of P w.r.t. the transformations rules $\mathcal{CS}_1$.

Definition 5 (D'-WFS)
Give a program P , we define D'-WFS(P) = $SEM_{min}(res_{\mathcal{CS}_1}(P))$.

We note that our definition of D'-WFS(P) is very similar to the definition of the D-WFS semantics (Brewka et al. 1997). The difference is that D-WFS defines pure disjunctions while D'-WFS(P) defines literals[7].

Let us note that although the $\mathcal{CS}_1$ system has the nice property of confluence (and termination), its computational properties are not so efficient. In fact, computing the residual form of a program is exponential (even for normal programs, whereas it is known that the WFS ((Gelder et al. 1991)) can be computed in quadratic time).

Definition 6 (Dloop)
(Arrazola et al. 1999) For a program P_1, let $unf(P_1) := \mathcal{L} \setminus MM(Definite(dis - nor(P_1))$. The transformation **Dloop(dp)** reduces a program P_1 to P_2:= $\{\mathcal{A} \leftarrow \mathcal{B}^+, \neg\mathcal{B}^- |\ \mathcal{B}^+ \cap unf(P) = \emptyset\}$. We assume that the given transformation takes place only if $P_1 \neq P_2$. We write $P_1 \longrightarrow_{Dloop_A} P_2$ to denote that P_1 transforms to P_2 by a **dp** transformation, where $A := unf(P_1)$.

Let **Dsuc** be the natural generalization of **suc** to disjunctive programs, formally:

Definition 7 (Dsuc)
(Arrazola et al. 1999) Suppose that P is a program that includes a fact $a \leftarrow true$ and a clause $\mathcal{A} \leftarrow Body$ such that $a \in Body$. Then we replace this clause by the clause $\mathcal{A} \leftarrow Body \setminus \{a\}$.

Definition 8 ($\mathcal{CS}_2$)
(Arrazola et al. 1999) Let $\mathcal{CS}_2$ be the rewriting system based on the transformations SUB, RED$^+$, RED$^-$, Dloop and Dsuc.

[7] The paper could be worked out without defining D'-WFS but using instead D-WFS.

Theorem 1
[Confl. and termination of $\mathcal{CS}_2$](Arrazola et al. 1999) The system $\mathcal{CS}_2$ is confluent and terminating. It induces a semantics that we call D1-WFS and define as D1-WFS(P)=$SEM_{min}(res_{\mathcal{CS}_2}(P))$. If we consider only normal programs then its induced semantics corresponds to the well-founded semantics.

D1-WFS generalizes a system introduced in (Brass et al. 1996) from normal to disjunctive programs.

Let us consider again Example 2. As we noticed before, program P reduces to P_3. But P_3 still reduces (by RED$^-$) to P_4, which is as P_3 but the third clause is removed. From P_4 we can apply a Dloop reduction to get P_5: the single clause $c \vee d \leftarrow true$. P_5 is the residual form of the $\mathcal{CS}_2$ system.

For this example it turns out that D'-WFS is equivalent to D1-WFS, but this is false in general. Take for instance, the following example:

Example 3
Let P be a disjunctive logic program:

$$\begin{array}{ll} \mathtt{x} \leftarrow \neg\mathtt{a}. & \mathtt{a} \leftarrow \neg\mathtt{b}. \\ \mathtt{d} \leftarrow \mathtt{a}. & \mathtt{c} \vee \mathtt{b}. \\ \mathtt{a} \leftarrow \mathtt{d}. & \mathtt{b} \leftarrow \mathtt{c}. \end{array}$$

Notice that D'-WFS(P)=$\{\mathtt{x}, \mathtt{b}, \neg\mathtt{d}, \neg\mathtt{c}, \neg\mathtt{a}\}$ but D1-WFS(P) = ϕ.

However for normal programs both systems are equivalent since they define WFS, but note that the residual programs w.r.t. $\mathcal{CS}_1$ and $\mathcal{CS}_2$ are not necessarily the same. An advantage of $\mathcal{CS}_2$ over $\mathcal{CS}_1$ is that the residual program w.r.t. $\mathcal{CS}_2$ is polynomial-time computable.

3 Declarative semantics

We first state and prove some useful properties.

Lemma 1
Let P be a program. If M is a minimal model of P and a supported model of P then M is a minimal supported model of P.

Proof
Our proof is by contradiction. Let M be a minimal model of P, M a supported model of P and M not a minimal supported model of P. Then exists a proper subset M' of M such that M' is a supported model of P. Thus M' is a model of P. But since M' is a proper subset of M, M is not a minimal model of P, obtaining a contradiction. □

The converse of this lemma is false as the following example shows:
$a \leftarrow \neg a.$
$a \leftarrow b.$
$b \leftarrow b.$
Note that $M := \{a, b\}$ is a minimal supported model of *dcomp*(P) but M is not a minimal model of P, and so it is not a minimal and supported model of P.

Lemma 2
If P_1 is a disjunctive program, $P_1 \longrightarrow_{Dloop_A} P_2$ then: $a \in A$ implies that for every clause $\alpha \leftarrow Body \in P_1$ such that $a \in \alpha$ then $Body \cap A \neq \emptyset$.

Proof
It is a straightforward generalization for the case in normal programs (Brass et al. 1996). □

Lemma 3
If $P_1 \longrightarrow_{Dloop_A} P_2$ then $Head(P_2) \cap A = \emptyset$.

Proof
By contradiction: Suppose $Head(P_2) \cap A \neq \emptyset$ therefore: exists $a \in Head(P_2) \cap A$. Then there is a rule of the form: $a \vee \alpha \leftarrow \beta \in P_2$. So $a \vee \alpha \leftarrow \beta \in P_1$. As $a \in A$ then exists $b \in \beta \cap A$ (by Lemma 2). Thus $a \vee \alpha \leftarrow \beta \notin P_2$, by the construction of P_2. This leads a contradiction. □

Lemma 4
Let T_1 and T_2 two theories then $T_1 \equiv_I T_2$ implies $T_1 \equiv_{stable} T_2$.

Proof
Follows immediately by Theorem 3.4 in (Pearce 1999) □

Lemma 5
If P is a theory, I a set of atoms and M any stable model of P, then $I \cap M = \emptyset \Rightarrow P \equiv_{stable} P \cup \neg I$

Proof
Follows immediately by Theorem 3.4 in (Pearce 1999) □

Lemma 6
(Brass and Dix 1997) Let P be program, if M is a stable model of P then M is a supported model of P.

Lemma 7
Let P_1 be a disjunctive program. If M is minimal model of P_1 and $P_1 \longrightarrow_{Dloop_A} P_2$ then $M \cap A = \emptyset$

Proof
Suppose $M \cap A \neq \emptyset$ then $b \in M$ and $b \in A$, so b is not in every minimal model of $definite(dir - nor(P))$, then $definite(dis - nor(P)) \nvdash b$, so $(dis - nor(P) \nvdash b$. Therefore $b \notin M$ for all M minimal model of $dis - nor(P)$, there after $b \notin M$ for all model of P then M is not minimal model. □

Osorio et al. in (Osorio et al. 2001) proved that stable semantics is closed under *Dloop*.

Theorem 2 (*Dloop preserve stable*)
(Osorio et al. 2001) Let P_1 be a disjunctive program. If $P_2 = Dloop(P_1)$ then P_1 and P_2 are equivalent under the stable semantics.

The following result was presented in (Osorio et al. 2001).

Lemma 8 (*STABLE is closed under $\mathcal{CS}_2$ transformations*)
(Osorio et al. 2001) Let P_1 and P_2 two programs related by any transformation in $\mathcal{CS}_2$. Then P_1 and P_2 have the same STABLE models.

Definition 9
Let S a set of literals and P be a program. We define $P^S as follows$:
$P^S := \{A \leftarrow \alpha \setminus (\alpha \cap S)$ such that $A \leftarrow \alpha \in P$ and it is false that there is $l \in \alpha$ and $l^c \in S\}$, where l^c is the complement of the literal l.

An interesting property of the $\mathcal{CS}_2$ system is the following:

Lemma 9
Let P_1 be a normal program and P_2 be the residual program resulting w.r.t. the $\mathcal{CS}_2$ system. Then $P_2 = P^{SEM_{min}(P_2)}$

Proof
First note that if P is obtained from P_1 by a single transformation $T \in \mathcal{CS}_2$ then $P^{SEM_{min}(P)} = P_1^{SEM_{min}(P)}$. So, by a direct induction, we can verify that $P_2^{SEM_{min}(P_2)} = P_1^{SEM_{min}(P_2)}$. But $P_2 = P_2^{SEM_{min}(P_2)}$. So, by transitivity we get the desired result. □

Lemma 10
If P and P_1 are two programs such that P_1 is obtained from P by any transformation T in $\mathcal{CS}_2$, then M is a supported model of P_1 implies M is supported model of P.

Proof
Straightforward by checking each case of T. □

The definition of our first semantics is:

Definition 10
Let P be a program. We define a D3-WFS-DCOMP model as a minimal model of P that also is a supported model of $res_{\mathcal{CS}_2}(P)$.

It is immediate to see that D3-WFS-DCOMP is more powerful than D1-WFS. Note that sometimes STABLE is inconsistent, when D3-WFS-DCOMP is not. Consider again the Example 1 where we saw that STABLE is inconsistent while D3-WFS-DCOMP defines exactly one model, which is $\{a, b\}$.

Lemma 11
Let P be a program. If M is D3-WFS-DCOMP model of P then M is a supported model of P.

Proof
Let M be D3-WFS-DCOMP model of P. So, M is a supported model of $res_{\mathcal{CS}_2}(P)$. Thus by Lemma 10 and a direct induction, M is a supported model of P. □

Due to its construction, we see that D3-WFS-DCOMP is similar to STABLE. However, STABLE is inconsistent more often than D3-WFS-DCOMP. This comment is formalized in the following theorem, which is one main result of this paper.

Theorem 3
Let P be a normal program.

1.-If M is a D3-WFS-DCOMP model of P, then M extends the WFS semantics (i.e. M agrees in the true/false assignments with WFS).

2.- If M is a D3-WFS-DCOMP model of P, then M is a minimal model of P as well as a minimal model of $comp(P)$.

3.- M is a STABLE model of P implies M is a D3-WFS-DCOMP model of P.

Proof
(1) follows by construction and Theorem 1. (2) follows by construction and lemmas 1 and 11. (3) Let M be a stable model of P. Hence (by Lemma 8) M is a stable model of $res_{\mathcal{CS}_2}(P)$. Hence by (Brass and Dix 1997), M is a supported model of $res_{\mathcal{CS}_2}(P)$. On the other hand, since M is stable model of P, it is well kwown that M is a minimal model of P. So M is a minimal model of P as well as a supported model of $res_{\mathcal{CS}_2}(P)$. Finally (by definition) M is a D3-WFS-DCOMP model of P.
□

We define our second semantics D2-WFS-DCOMP.

Definition 11
Let P be a program. We define a D2-WFS-DCOMP model as a minimal model of P that also is a supported model of $P^{D'-WFS(P)}$.

Lemma 12
The semantics D3-WFS-DCOMP is not equivalent to D2-WFS-DCOMP.

Proof
Consider again our Example 3. With D3-WFS-DCOMP we obtain two models: $\{\mathtt{a}, \mathtt{b}, \mathtt{d}\}$ and $\{\mathtt{b}, \mathtt{x}\}$. However, with D2-WFS-DCOMP we only obtain the model $\{\mathtt{b}, \mathtt{x}\}$.
□

We conjecture the following: for every program P, every D2-WFS-DCOMP model of P is a D3-WFS-DCOMP model of P.

We also suggest to study the following semantics. The motivation is to enforce the interpretation of $\vee$ as inclusive disjunction. In (Greco 1999) the authors also define a semantics based on this notion.

Definition 12
Let P be a program, a w-supported model M of P is a model of P such that for every $a \in M$, there is a clause $\mathcal{A} \leftarrow body$ such that $a \in \mathcal{A}$ and the *body* is true in M.

Definition 13
Let P be a program. We define a D3-WFS1-DCOMP model as a minimal model of P that also is a w-supported model of $P^{D1-WFS(P)}$

Theorem 4
Let P be any normal program, then D3-WFS-DCOMP(P)=D2-WFS-DCOMP(P)=D3-WFS1-DCOMP(P).

Proof
Since $D1-WFS(P) = D'-WFS(P)$ (for normal programs) and by Lemma 9 then is immediate that D3-WFS-DCOMP(P)=D2-WFS-DCOMP(P). Since supported models are the same as w-supported models (for normal programs) then D3-WFS-DCOMP(P)=D3-WFS1-DCOMP(P).
□

Note that however, for disjunctive programs D3-WFS-DCOMP is (in general) different to D1-WFS1-DCOMP. Take for instance, Example 2 in (Greco 1999).

4 Extensions

We sketch how to extend our semantics to consider programs with clauses allowing an empty head, explicit negation, datalog programs and finally a *lub* operator.

- With respect to datalog programs, we first obtain the ground instantiation of the program and then we proceed as with propositional programs. This is a well known approach and we do not discuss it any more.
- With respect to programs with explicit negation, it is possible to reduce programs with explicit negation to programs without it. The idea is originally considered in (Baral and Gelfond 1994).
- With respect to programs that include clauses with empty head we proceed as follows: Say that we have the clause:
  $\leftarrow \ \alpha$ then simply translate it as: $a \leftarrow \ \alpha, \ \neg a$
  where a is a new atom. Clearly, α is false in every D3-WFS-DCOMP model of the program.

4.1 Semantics of lub-programs

A very interesting issue consists in modeling the *setof* operator of PROLOG. In a meeting, whose aim was to get a clear picture of the current activities in Logic Programming and its role in the coming years, Gelfond pointed out: "We need to find elegant ways for accomplishing simple tasks, e.g. counting the numbers of

elements in the database satisfying some property P. At the moment we have neither clear semantics nor efficient implementation of *setof* of other aggregates used for this type of problems" (Dix 1998). We consider the following example.

```
s(S) ← setof(X, p(X), S).
```

where the intended meaning is: `s(S)` is true when $\mathtt{S} = \{\mathtt{X} : \mathtt{p}(\mathtt{X})\}$.

Several authors have suggested to translate the above expression using negation as failure more or less as follows:

```
s(S) ← setof-s(S).
setof-s(S) ← try-s(S),¬ bad-s(S).
try-s(S) ← p(X), singleton-set(X,S).
try-s(S) ← try-s(S1), try-s(S2), union(S1,S2,S).
try-s(X) ← emptyset(X).
bad-s(S) ← p(X), ¬ member(X,S).
```

where union, member and singleton-set have their intended meaning. For instance, `singleton-set(X,S)` is true if $\mathtt{S} = \{\mathtt{X}\}$. Given any normal program extended with *setof*, if the translation is a stratified program, then the stratified model of the program captures the "intended" semantics of the original program (Osorio and Jayaraman 1999). However, when the program is not stratified then we can consider using stable models. Several times the stable models still captures the intended meaning but sometimes it becomes undefined.

Here is a simple but yet an interesting example about *setof*. Consider the definite program:

$\mathtt{a} \leftarrow \mathit{true}.$

$\mathtt{b} \leftarrow \mathtt{a}, \mathtt{c}.$

represented as:

```
rule([a]).
rule([b,a,c]).
```

That is, `rule([X|Y])` represents the clause $\mathtt{X} \leftarrow \mathtt{Y}$. Think that we want to write a program that computes the minimal model. Thinking in the fix-point semantics we could write:

```
mm(S) ← setof(X, t-p(X), S).
t-p(X) ← rule(X,Y), mm(S), subset(Y,S).
```

That is, `mm(S)` is true (in the intended semantics) iff `S` is the minimal model of the program represented by the EDB (extensional database) `rule`[8]. Note that `mm` and `t-p` are mutually dependent. If we use the translation of *setof* to NF given above, then STABLE has no models at all, while any of our proposed semantics in this paper define the intended model.

The ideas of this part are taken from (Osorio and Jayaraman 1999) but adapted to the more familiar environment of PROLOG. A `LUB` operator is a generalization of *setof* over any complete lattice. The intuitive reading is that $\mathtt{LUB}_L\mathtt{(X, p(X), S)}$ is true if `S` is the least upper bound of each `X` such that `p(X)`.

Let `P` be a logic program:

[8] It is however questionable if the above program should be considered legal.

$$\texttt{h(X,S)} \quad \leftarrow \quad \texttt{LUB}_\texttt{L}\texttt{(C,n(X,C),S)}.$$

Now translate the program before as follows:

$$\begin{array}{lll} \texttt{h(X,S)} & \leftarrow & \texttt{f}_{=}\texttt{(X,S).} \\ \texttt{f}_{\geq}\texttt{(X,S)} & \leftarrow & \texttt{n(X,S).} \end{array}$$

In addition we need to add axioms that relate the predicate symbols $\texttt{f}_{=}$ with $\texttt{f}_{\geq}$ for each functional symbol f. Let us consider again our example. The axioms for f in this case are as follows:

(1) $\texttt{f}_{=}\texttt{(Z, S)} \leftarrow \texttt{f}_{\geq}\texttt{(Z,S)}, \ \neg\, \texttt{f}_{>}\texttt{(Z, S).}$

(2) $\texttt{f}_{>}\texttt{(Z, S)} \leftarrow \texttt{f}_{\geq}\texttt{(Z,S1)}, \ \texttt{S1} >_\texttt{L} \texttt{S.}$

(3) $\texttt{f}_{\geq}\texttt{(Z, S)} \leftarrow \texttt{f}_{\geq}\texttt{(Z,S1)}, \ \texttt{S1} >_\texttt{L} \texttt{S.}$

(4) $\texttt{f}_{\geq}\texttt{ (Z, } \bot\texttt{).}$

(5) $\texttt{f}_{\geq}\texttt{(Z,C)} \leftarrow \texttt{f}_{\geq}\texttt{(Z,C}_1\texttt{)}, \ \texttt{f}_{\geq}\texttt{(Z,C}_2\texttt{)}, \ \texttt{lub}_\texttt{L}\texttt{(C}_1\texttt{,C}_2\texttt{,C).}$

We understand that $\texttt{S1} >_\texttt{L} \texttt{S}$ means that $\texttt{S1} \geq_L \texttt{S}$ and $\texttt{S1} \neq \texttt{S}$. And $\texttt{lub}_\texttt{L}\texttt{(C}_1\texttt{,C}_2\texttt{,C)}$ interprets that C is the least upper bound of C_1 and C_2. The first two clauses are the same (modulo notation) as in definition 4.2 in (Gelder 1992). Clause (5) is not useful for total-order domains.

Now we need to instantiate the translated program. In this paper we restrict our attention to finite ground programs. By adding simple type declarations we can ensure to get a finite ground program. In this case, we borrow the declaration style of Relationlog, see (Osorio and Jayaraman 1999).

5 Conclusion and future work

We presented some useful results about the stable semantics STABLE. We propose three semantics as alternative options for STABLE. The aim of our proposals is a solution of some drawbacks of disjunctive (and normal) stable semantics that becomes undefined (inconsistent) very often. We also sketch some extensions, mainly to allow at *lub* operator, a general form of *setof*. We noticed that STABLE could not define the intended meaning of a simple program that uses our *lub* operator.

In (Nieves et al. 2005), we use stable models semantics in order to model the Dung's argumentation approach(Dung 1995) and also we present some extensions of the Dung's argumentation semantics. One of our future work is to explore the application of the semantics presented in this paper in argumentation theory. For instance in argumentation theory, there is a few work *w.r.t.* aggregation of arguments, we believe that the axiomatization presented in Section 4.1 could be helpful in order to handled aggregation between arguments. Moreover in (Nieves and Cortés 2006), we present some preliminary results, *w.r.t.* to use a similar axiomatization of the Section 4.1 in order to infer the certain of an argument.

References

Arrazola, J., Dix, J., and Osorio, M. 1999. Confluent rewriting systems in non-monotonic reasoning. *Computación y Sistemas 2,* 2-3, 104–123.

Baral, C. and Gelfond, M. 1994. Logic programming and knowledge representation. *J. Log. Program. 19/20*, 73–148.

Brass, S. and Dix, J. 1997. Characterizations of the disjunctive stable semantics by partial evaluation. *J. Log. Program. 32,* 3, 207–228.

Brass, S. and Dix, J. 1998. Characterizations of the disjunctive well-founded semantics: Confluent calculi and iterated gcwa. *J. Autom. Reasoning 20,* 1, 143–165.

Brass, S., Zukowski, U., and Freitag, B. 1996. Transformation-based bottom-up computation of the well-founded model. In *NMELP*. 171–201.

Brewka, G., Dix, J., and Konolige, K. 1997. *Nonmonotonic Reasoning : An overview.* CSLI Lectures Notes 73. CSLI Publications, Stanford, CA.

Dix, J. 1995. A classification theory of semantics of normal logic programs: Ii. weak properties. *Fundam. Inform. 22,* 3, 257–288.

Dix, J. 1998. The logic programming paradigm. *AI Commun. 11,* 2, 123–131.

Dix, J., Osorio, M., and Zepeda, C. 2001. A General Theory of Confluent Rewriting Systems for Logic Programming and its applications. *Annals of Pure and Applied Logic 108,* 1–3, 153–188.

Dung, P. M. 1992. On the relations between stable and well-founded semantics of logic programs. *Theor. Comput. Sci. 105,* 1, 7–25.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence 77,* 2, 321–358.

Gelder, A. V. 1992. The well-founded semantics of aggregation. In *PODS*. 127–138.

Gelder, A. V., Ross, K. A., and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38,* 3, 620–650.

Gelfond, M. and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *5th Conference on Logic Programming*, R. Kowalski and K. Bowen, Eds. MIT Press, 1070–1080.

Greco, S. 1999. Minimal founded semantics for disjunctive logic programming. In *LPNMR*. 221–235.

Lloyd, J. W. 1987. *Foundations of Logic Programming.* Springer, Berlin.

Nieves, J. C. and Cortés, U. 2006. Modality argumentation programming. In *Proceedings of the conference Modeling Decisions for Artificial Intelligence (MDAI 2006) (To appear).* Springer LNAI.

Nieves, J. C. and Jayaraman, B. 2002. Relaxation in partial order programming. In *Workshop on Logic and Computation: MICAI'2002.*

Nieves, J. C., Osorio, M., and Cortés, U. 2005. Modeling argumentation based semantics using non-monotonic reasoning. Tech. rep., Universitat Politècnica de Catalunya, Software Department, Barcelona, Spain. December.

Osorio, M. and Jayaraman, B. 1999. Relating aggregation and negation-as-failure. *New Generation Comput. 17,* 3, 255–284.

Osorio, M., Jayaraman, B., and Plaisted, D. A. 1999. Theory of partial-order programming. *Sci. Comput. Program. 34,* 3, 207–238.

Osorio, M., Navarro, J. A., and Arrazola, J. 2001. Equivalence in Answer Set Programming. In *Logic Based Program Synthesis and Transformation, 11th International Workshop, LOPSTR 2001, Paphos, Cyprus, November 28-30.* 57–75.

Osorio, M. and Zacarias, F. 2000. High-level logic programming. In *FoIKS*. 226–240.

Pearce, D. 1999. Stable Inference as Intuitionistic Validity. *Logic Programming 38*, 79–91.

Przymusinski, T. C. 1991. Well-founded completions of logic programs. In *ICLP*. 726–741.

You, J.-H. and Yuan, L.-Y. 1994. A three-valued semantics for deductive databases and logic programs. *J. Comput. Syst. Sci. 49,* 2, 334–361.