Computación y Sistemas Vol. 2 Nos. 2-3 pp. 104-113
© 1999, CIC - IPN. *ISSN 1405-5546* Impreso en México

Confluent Rewriting Systems in Non-Monotonic Reasoning

Jose Arrazola
Benemerita Universidad
Autonoma de Puebla
Facultad de Ciencias
Fisico Matematicas
75579 Puebla, Mexico

Jürgen Dix
University of Koblenz
Department of
Computer Science
Rheinau 1, D-56075
Koblenz, Germany
dix@informatik.uni-koblenz.de

Mauricio Osorio
Universidad de las Americas
Ing. en Sist. Computacionales
Sta. Catarina Martir,
Cholula, Puebla
72820 Mexico
josorio@mail.udlap.mx

Article received on October 12, 1998; accepted on February 25, 1999

Abstract

We introduce the general notion of a Confluent LP-System, *which is a rewriting system on the set of all logic programs over a signature $\mathcal{L}$. Such a system is based on certain transformation rules and induces a semantics SEM in a natural way. We show that most of the well-known semantics for normal logic programs are induced by confluent LP-systems. Moreover, we show by introducing several new transformation rules that the corresponding LP-systems induce interesting semantics which are polynomial time computable and extend WFS. Moreover we use our approach to define new semantics for disjunctive programs.*

Keywords:

Well-founded semantics, stable semantics, logic programming, non-monotonic reasoning, rewriting systems, negation as failure.

1 Introduction

In this paper we try to combine methods from rewriting with logic programming technology to get a framework for considering semantics of logic programs.

The main idea (already introduced in [Brass and Dix, 1998] for disjunctive logic programs) is to determine a set of rewriting rules that is confluent. These rules transform programs into *simpler* programs. Confluence and termination guarantees that every program is associated a *normal form*. This normal form then induces a semantics and a simple and efficient method to answer queries with respect to this semantics.

In this paper we do not stick to a particular semantics but we develop the beginnings of a general theory of confluent LP systems and investigate which semantics can be represented as such systems.

We also extend the confluent system which corresponds to the wellfounded semantics WFS by new rewriting rules. These rules are motivated by the use of *aggregates* in logic programming and the problems of modelling aggregates in semantics such as WFS (see [Osorio and Jayaraman, 1997; Dix and Osorio, 1997]).

An advantage of our approach is its declarative nature. The transformation rules express natural conditions and different applications might ask for different selections of such rules. We show that the naturally induced semantics of a confluent LP-system is in fact the *weakest* semantics satisfying all our transformations.

The first proposal to provide a convincing declarative semantics to NAF was given in [Clark, 1978] and it is called the Clark's completion. The main idea is that, to deduce negative information from a normal program, we could "complete" the program by adding the only-if halves of the definitions of the predicate symbols[1]. Clark's completion is obtained as our weakest LP-system $\mathcal{CS}_0$

[1]For the details we refer to [Lloyd, 1987].

However, it is now accepted that Clark's completion is often too weak and does not always capture the intended meaning of logic programs necessary for e.g Knowledge Representation tasks [Baral and Gelfond, 1994; Brewka and Dix, 1996]. Therefore stronger semantics are needed.

The two main competing approaches are the *well founded* [Gelder *et al.*, 1988, and 1991] and the *stable* semantics [Gelfond and Lifschitz, 1988, and 1990]. WFS has received great interest since its introduction, because it has several desirable properties: it is always consistent and every program possesses exactly one 3-valued model. In addition, this model is computable in polynomial time.

In contrast to this, the stable semantics is not always consistent and answering a query does not depend on the call-graph below that query. This implies that there is no *goal-directed* computation possible. In addition, computing stable models is NP-hard and can therefore not be done in polynomial time (unless the polynomial hierarchy collapses).

One of the problems of WFS is that it is overly sceptical: clauses of the form $a \leftarrow \neg a$ are treated as *undefined*. This is particurlarly bad when aggregate predicates are considered (where a should be treated as *true*). We introduce in this paper several confluent LP-systems to solve the problems related with $a \leftarrow \neg a$ clauses. Some of them are polynomial time computable and satisfy also *rationality* [Dix, 1995a].

Our definitions and results are given for propositional programs only, to ensure termination. But we believe that if we drop termination and only retain confluence, we can get similar results for general programs with variables (under the assumption that the normal form exists). We also apply our approach to define new semantics for disjunctive programs.

Our paper is structured as follows. After giving some background information on the concepts involved (section 2), we introduce in section 3 our new notion of a *Non-monotonic confluent LP-system*. Our main result in this section is that *partial distribution* implies *rationality*. Section 4 shows that most of the well known semantics can be defined in a natural way via confluent LP-systems. Section 5 introduces several confluent LP-systems that define new rational and polynomial-time computable semantics. Section 6 introduces new semantics for disjunctive programs.

2 Background

We first collect some general results about rewriting systems that are needed later. We then define the notion of SEM_{min}, stating the minimal requirements any semantics should have.

Definition 2.1 (Rewriting System)
An abstract rewriting system is a pair $\langle S, \rightarrow \rangle$ where $\rightarrow$ is a binary relation on S. Let $\rightarrow^*$ be the reflexive, and transitive closure of $\rightarrow$. When $x \rightarrow^* y$ we say that x reduces to y. *An* irreducible *element is said to be in* normal form.

We say that a rewriting system is

noetherian: *if there is no infinite chain $x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_i \rightarrow x_{i+1} \rightarrow \ldots$*

confluent: *if whenever $u \rightarrow^* x$ and $u \rightarrow^* y$ then there is a z such that $x \rightarrow^* z$ and $y \rightarrow^* z$.*

locally confluent: *if whenever $u \rightarrow x$ and $u \rightarrow y$ then there is a z such that $x \rightarrow^* z$ and $y \rightarrow^* z$.*

In a noetherian and confluent rewriting system, every element x reduces to a unique normal form that we denote by $norm(x)$.

A signature $\mathcal{L}$ is a finite set of elements that we call *atoms*. A *literal* is an atom or the negation of an atom a that we denote by $\neg a$. Given a set of atoms $\{a_1, \ldots, a_n\}$, we write $\neg\{a_1, \ldots, a_n\}$ to denote $\{\neg a_1, \ldots, \neg a_n\}$. We may denote a normal clause C as usual [Lloyd, 1987]: a :- $l_1, \ldots, l_n$, where a is an atom and each l_i is a literal; or by $a \leftarrow \mathcal{B}^+, \neg\mathcal{B}^-$, where $\mathcal{B}^+$ contains all the positive body atoms and $\mathcal{B}^-$ contains all the negative body atoms. We also use $body(C)$ to denote $\mathcal{B}^+ \cup \neg\mathcal{B}^-$. A program is a finite set of clauses. Sometimes we will consider the logical constants **t** and **f** with their intended interpretation. Let $Prog_{\mathcal{L}}$ be the set of all normal propositional programs with atoms from $\mathcal{L}$. By $\mathcal{L}_P$ we understand the signature of P, i.e. the set of atoms that occur in P. A (partial) interpretation based on a signature L is a disjoint pair of sets $\langle I_1, I_2 \rangle$ such that $I_1 \cup I_2 \subseteq \mathcal{L}$. A partial interpretation is total if $I_1 \cup I_2 = \mathcal{L}$. Given two interpretations $I = \langle I_1, I_2 \rangle$, $J = \langle J_1, J_2 \rangle$, we define $I \leq_k J$ iff $I_i \subseteq J_i$, $i = 1, 2$. Clearly $\leq_k$ is a partial order. We may also see an interpretation $\langle I_1, I_2 \rangle$ as the set of literals $I_1 \cup \neg I_2$. When we look at interpretations as sets of literals then $\leq_k$ corresponds to $\subseteq$.

A general semantics SEM is a function on $Prog_{\mathcal{L}}$ which associates to every program a partial interpretation.

What are the minimal requirements we want to impose on a semantics? Certainly we want that facts, i.e. rules with empty bodies are true. Dually, if an atom does not occur in any head, then its negation should be true. This gives rise to the following definition (given by Brass and Dix), which will play an important role later.

Definition 2.2 (SEM_{min})
For any program P we define $HEAD(P) = \{a |\ a \leftarrow \mathcal{B}^+, \neg\mathcal{B}^- \in P\}$ — the set of all head-atoms of P. We also define
$SEM_{min}(P) = \langle P^{true}, P^{false} \rangle$ *, where*
$P^{true} := \{p |\ p \leftarrow \in P\}$, *and*

$P^{false} := \{p |\ p \in \mathcal{L}_P \backslash HEAD(P)\}$

3 Non-monotonic confluent LP-Systems

The main concept our notion of a LP system is based upon, is the concept of a *transformation* rule, see [Brass and Dix, 1997].

Definition 3.1 (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$^+$: *This transformation can be applied to P, if there is an atom a which does not occur in HEAD(P). RED$^+$ transforms P to the program where all ocurrences of $\neg a$ are removed.*

RED$^-$: *This transformation can be applied to P, if there is a rule $a \leftarrow \ \in P$. RED$^-$ transforms P to the program where all clauses that contain $\neg a$ in their bodies are deleted.*

SUB: *This transformation can be applied to P, if P contains two clauses $a \leftarrow body_1$, and $a \leftarrow body_2$, where $body_1 \subseteq body_2$. SUB transforms P to the program where the clause $a \leftarrow body_2$ has been removed.*

Although these rules are not really functions *on $Prog_{\mathcal{L}}$ (e.g. RED$^-$ is only determined if an occurrence of a certain rule is distinguished), we usually write them as operators on $Prog_{\mathcal{L}}$ when it is understood by context how they are uniquely determined.*

Obviously, the just mentioned transformations are among the minimal requirements a *well-behaved* semantics should have (see [Dix, 1995b]). From now on, when we speak of a semantics SEM, we understand that SEM is invariant under the transformations RED$^+$, RED$^-$ and SUB.

Given a program P and a list of operators *ops*, we define the application of *ops* to P, denoted as P^{ops}, as follows:

$$\begin{array}{lcl} P^{[]} & := & P \\ P^{[op|ops]} & := & (P^{op})^{ops} \end{array}$$

We define the size of *ops* as the number of elements of *ops*. We are now ready to introduce our main notion:

Definition 3.2 (Confluent LP-system $\mathcal{CS}$)
A non-monotonic confluent LP-system *$\mathcal{CS}$ over the signature $\mathcal{L}$ is a pair*

$$\langle \{op_i \mid i = 1, \ldots, n\}, \rightarrow \rangle$$

that satisfies the following conditions:

1. *$\{op_i \mid i = 1, \ldots, n\}$ is a finite set of transformation rules on $Prog_{\mathcal{L}}$. By abuse of language we often view them as operators and we write P^{op} to denote the result of the application of the given operator. We say that the operator is* executed *in this case.*

2. *$\langle Prog_{\mathcal{L}}, \rightarrow \rangle$, where $\rightarrow$ is the union of all the transformation rules in $\mathcal{CS}$, is a* noetherian *and* confluent *rewriting system.*

3. *If $P \rightarrow P_1$ then $SEM_{min}(P) \leq_k SEM_{min}(P_1)$.*

We denote the uniquely determined normal form of a program P with $norm_{\mathcal{CS}}(P)$.

Every LP-system $\mathcal{CS}$ induces a semantics $SEM_{\mathcal{CS}}$ as follows:

$$SEM_{\mathcal{CS}}(P) := SEM_{min}(norm_{\mathcal{CS}}(P))$$

When there is no ambiguity about the given confluent LP-system we drop the subscripts.

Some points are worth mentioning:

1. Thanks to confluence and termination, our transformation rules have both a declarative and an operational meaning. From the declarative point of view, they tell us that our semantics is closed under the given transformation rule. From an operational point of view the transformations are computable functions that can be applied to *simplify* the program. When we arrive to the normal form, then computing its semantics is immediate.

2. Rationality (introduced later) has been argued as a desirable property of semantics in logic programming. Confluence plus a simple property that we call *partial distribution* implies rationality. It is straightforward to see that all (but one) of our semantics satisfy partial distribution. Therefore we know that these semantics are rational.

3. The nature of simple transformations, confluence and termination allow us to prove many properties by induction on the number of steps that we need to compute the normal form.

3.1 Basic Results

We omitt the proofs of most of our results but they can be found in the research report [Dix *et al.*, 1997].

Lemma 3.1 *For every* confluent LP-system *$\mathcal{CS}$ and every program P such that every step in P^{ops} is executed we have: $SEM_{min}(P) \leq_k SEM_{min}(P^{ops})$.*

Corollary 3.1
For every program P of a confluent LP-system *$\mathcal{CS}$: $SEM_{min}(P) \leq_k SEM_{\mathcal{CS}}(P)$.*

The following condition is also desirable in logic programming.

Definition 3.3 (Three-valued Based)
A semantics SEM is called a 3-valued based *if for every program P the partial interpretation SEM(P) is a 3-valued model of P.*

In 3-valued based semantics, it can not happen that the body of a rule evaluates to *true* in SEM(P) while the head of this rule evaluates to *false* or to *undefined*. For the details of 3-valued based semantics see [Dix, 1995a; Brass and Dix, 1997].

3.2 Rationality

A semantics SEM is called *rational*, if for every atom a and program P the following holds

$$\neg a \notin \mathrm{SEM}(P) \text{ implies } \mathrm{SEM}(P) \leq_k \mathrm{SEM}(P \cup \{a \leftarrow\}).$$

Definition 3.4 (Partial distribution)
A confluent LP-system $\mathcal{CS}$ *satisfies* partial distribution *if for every op, P and a such that $a \in HEAD(P^{op})$ and P^{op} is executed, the following holds:*

1. *$(P \cup \{a\})^{op}$ is executed,*
2. *$(P \cup \{a\})^{op} = P^{op} \cup \{a\}$.*

Lemma 3.2 *For every partial distributive* confluent LP-system $\mathcal{CS}$ *the following is true. Let $a \in HEAD(P^{ops})$, where ops is any list of operators such that every step in P^{ops} is executed. Then:*

1. *every step in $(P \cup \{a\})^{ops}$ is executed,*
2. *$(P \cup \{a\})^{ops} = P^{ops} \cup \{a\}$.*

Rationality is a nice property for a semantics SEM, especially if we plan to use the system in logic programming. The following result is important because it shows that a relatively simple condition, namely partial distribution on $\mathcal{CS}$ suffices to ensure rationality of the induced semantics.

Theorem 3.1 (Rational semantics)
Every partial distributive confluent LP-system $\mathcal{CS}$ *induces a rational semantics $SEM_{\mathcal{CS}}$.*

4 Clark's completion, WFS and WFS$^+$

In this section we show that the 3-valued version of clark's completion semantics (introduced by Fitting in [Fitting, 1985, and 1986]), WFS ([Gelder *et al.*, 1988, and 1991]) and an extension of WFS (introduced independently by [Dix, 1992] and [Schlipf, 1992]) are induced by *confluent LP-system*.

To this end, we introduce the following transformation rules:

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

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

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

TAUT: *(Tautology)* Suppose P contains a rule C which has the same atom in its head and in its body. Then we can remove this rule.

LC: *(Logical Consequence)* Suppose $P \models a$ for an atom a. Then we can add the rule $a \leftarrow$ to P.

Let $\mathcal{CS}_0$ be the *LP-system* which contains, besides the basic transformation rules, the rule GPPE. Let $\mathcal{CS}_1$ be $\mathcal{CS}_0$ enlarged by TAUT. Finally, let $\mathcal{CS}_2$ be $\mathcal{CS}_1$ plus the rule LC. By using results of [Dix, 1995a; Brass and Dix, 1998] we get

Theorem 4.1 (Classifying comp$_3$, WFS)

1. *Fitting's semantics comp$_3$ is the weakest 3-valued based semantics satisfying GPPE. It is induced by the* confluent LP-system $\mathcal{CS}_0$.
2. *The wellfounded semantics is the weakest 3-valued based semantics satisfying GPPE and TAUT. It is induced by the* confluent LP-system $\mathcal{CS}_1$.

Let us note that although the $\mathcal{CS}_1$ system has the nice property of confluence (and termination), its computational properties are not that efficient. In fact, computing the normal form is exponential, whereas it is known that the WFS can be computed in quadratic time. It turned out that recently an equivalent confluent LP-system was defined, where the normal form can be computed in quadratic time (see [Brass *et al.*, 1997]). The idea is to restrict the GPPE rule and to replace it by two simple instances plus an additional check for an unfounded set (Loop detection):

Success (S): Suppose that P includes a fact a and a clause $q \leftarrow Body$ such that $q \in Body$. Then we replace the clause $q \leftarrow Body$ by $q \leftarrow Body \setminus \{a\}$.

Failure (F): Suppose that P includes an atom a and a clause $q \leftarrow Body$ such that $a \notin HEAD(P)$ and a is a positive literal in $Body$. Then we erase the given clause.

Loop Detection (Loop): We say that P_2 results from P_1 by $Loop_A$ iff there is a set A of atoms such that

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

We call the resulting system $\mathcal{CS}_{1'}$, i.e. **RED$^+$+RED$^-$+S+F+Loop**. We now show that **F** is redundant, for that we need the following lemma.

Lemma 4.1 (Zepeda, 1997)
Let $\mapsto_X$ denotes the reduction relation of the $\mathcal{CS}_{1'}$ system. Let $\mapsto_{X-F}$ be as $\mapsto_X$ without **F**. *For all programs P, P', P_1 , P_2, where,*

$$P \mapsto^n_{X-F} P' \mapsto_F P_1 \mapsto^*_X P_2$$

there is a program P'' such that

$$P \mapsto^n_{X-F} P' \mapsto_L P'' \mapsto^*_X P_2$$

Proof: If $P' \mapsto^n_F P_1$, where a is the atom resposible of this failure reduction. Let **A**=$\{a\}$. Suppose that there are exactly k clauses in P' such that a occurs positively in them. Starting with P' we can apply a sequence of **k** Failure steps (over the same positive literal) leading to some program P'' where a no longer occurs positively. And by confluence we get that,

$$P \mapsto^n_{X-F} P' \mapsto_F \cdots \mapsto_F P'' \mapsto^*_X P_2$$

Moreover: There is no rule $A \leftarrow \mathbf{B}$ in P' such that $A \notin \mathbf{A}$. $P'' := \{A \leftarrow \mathbf{B} \in P' \mid \mathbf{B} \cap \mathbf{A} = \emptyset\}$, and $P' \neq P''$. For hence P'' also results from P' by applying the *Loop Detection*, so

$$P \mapsto^n_{X-F} P' \mapsto_L P'' \mapsto^*_X P_2$$

as desired.
The following theorem is a direct consequence of the above lemma.

Theorem 4.2 (Elimination of Failure, Zepeda,97)

Let P be a program. Then P' the normal form of P w.r.t. the $\mathcal{CS}_{1'}$ system, iff P' is the normal form of P w.r.t the same system but without **F**.

Theorem 4.3 (Classifying WFS$^+$)
The WFS$^+$ semantics is the weakest 3-valued based semantics satisfying S, F, Loop, TAUT and LC. It is induced by the confluent LP-system *consisting of exactly these transformations (plus the basic ones).*

The reason we gave up GPPE is not only for computational purposes. As has been shown in [Dix and Müller, 1994] GPPE does not hold for WFS$^+$. Let us consider the programs:

$$P: \begin{array}{lll} a & \leftarrow & \neg b \\ b & \leftarrow & \neg a \\ x & \leftarrow & a \\ x & \leftarrow & b \end{array} \qquad P': \begin{array}{lll} a & \leftarrow & \neg b \\ b & \leftarrow & \neg a \\ x & \leftarrow & \neg a \\ x & \leftarrow & \neg b \end{array}$$

By applying GPPE twice, we get P'. But WFS$^+(P) = \{x\}$ while WFS$^+(P') = \emptyset$. We point out that the WFS for both P and P' is $\emptyset$, considered as a shortcoming of WFS.

5 New Semantics extending WFS

In this section we show how our transformation rules can be extended by new rules that still define confluent LP-systems.

The motivation of these rules is the overly sceptical approach of WFS. In particular for aggregation predicates, rules of the form $a \leftarrow \neg a$ should be considered as being equivalent to a. However, incorporating this into a well-behaved semantics is in general very difficult.

Of course the WFS$^+$ semantics has this property. But again it can be shown that WFS$^+$ is on the second level of the polynomial hierarchy (due to the LC-rule) and thus most probably not polynomial. We therefore introduce a weaker form of LC:

Definition 5.1 (Local Logical Consequence)
*By the application of LLC (*local logical consequence*) to a program P that only contains one clause with head a, namely a LC-clause $a \leftarrow \neg a \in P$, we mean the transformation of P which simply removes every occurrence of $\neg a$ in P.*

We define $\mathcal{CS}_3$ as LP-system $\mathcal{CS}_1$ plus the rule LLC. Here is one of our main theorems.

Theorem 5.1 ($\mathcal{CS}_3$ is a confluent LP-System)
$\mathcal{CS}_3$ is a confluent LP-System. Its induced semantics, called WFS^0, is 3-valued based, but not rational.

Here are stronger versions of LLC.

Definition 5.2 (LLC*)
Let B be an ordered set $\{a_0, \ldots, a_n\}$, where $n \geq 0$. Suppose in addition that the set of clauses $LLC_B := \{a_1 \leftarrow \neg a_0\} \cup \{a_{i+1} \leftarrow a_i : 1 \leq i \leq n-1\} \cup \{a_0 \leftarrow a_n\}$ is a subset of a program P. If $n = 0$ we assume that the set reduces to $a_0 \leftarrow \neg a_0$. Then the transformation $\langle LLC^*, B\rangle$ substitutes the clause $a_0 \leftarrow a_n$ by the fact a_0.

Note that by transitivity of $\leftarrow$ we get $a_0 \leftarrow \neg a_0$. Moreover $(\neg a \rightarrow a) \rightarrow a$ is a theorem in propositional classical logic and by *Modus Ponens* we can infer a. So, this rule is sound in propositional classical logic. In [Osorio *et al.*, 1995] it is defined a language that allows to model aggregation in a natural way via *POL* programs. In [Osorio and Jayaraman, 1997] we showed that this

POL programs can be translated to normal programs such that the declarative semantics of a POL program corresponds to the WFS$^+$ of the translated normal program. But we have recently noted, [Dix and Osorio, 1997], that not always we needed the full power of WFS^0 but a restricted form that corresponds to LLC*.

Let $\mathcal{CS}_4$ be the LP-system $\mathcal{CS}_{1'}$ plus the rule LLC*.

Theorem 5.2 (WFS + LLC*)
$\mathcal{CS}_4$ is a confluent LP-System. Its induced semantics is 3-valued based and rational.

Definition 5.3 (Contra)
Let C a clause of the form $a \leftarrow Body$, where both $b \in Body$ and $\neg b \in Body$. We define the transformation Contra as the one that deletes the clause C from the program P.

At this point we are really not sure about the potencial uses of *Contra*. The motivation for *Contra* comes from the following observations. Clearly the rule is a theorem in intuitionistic propositional logic and so is in classical logic. The power of a theory should not be affected if we add or delete theorems from the proper axioms of the theory (the proper axioms correspond to our programs).

Theorem 5.3 (WFS + Contra)
The LP-system given by $\mathcal{CS}_{1'}$ plus the rule Contra is confluent. In addition its induced semantics is rational but not 3-valued based.

It has been criticized the WFS for its inability to do reasoning by cases and this is one reason to prefer in several cases STABLE over WFS, see for instance [Baral and Gelfond, 1994]. The following rule allows to add a weak form of this kind of reasoning to WFS, still preserving the polynomial-time computability property of the semantics.

Definition 5.4 (Weak-Cases)
Let C_1 be of the form $a \leftarrow b$ and C_2 be of the form $a \leftarrow \neg b$, where $a \neq b$ are any pair of atoms of the signature. We define the transformation $\langle Weak-Cases, C\rangle$ as the one that substitutes the pair of clauses C_1, C_2 by the fact a in the program P.

Theorem 5.4 (WFS + Weak-Cases)
The LP-system given by $\mathcal{CS}_{1'}$ plus the rule Weak-Cases is confluent. In addition its induced semantics is rational and 3-valued based.

6 Disjunctive programs

We may denote a (general) clause C as: $a_1 \vee \ldots \vee a_m$:- $l_1, \ldots, l_n$, where $m > 0$, $n \geq 0$, each a_i is an atom, and each l_i is a literal. When $n = 0$ the clause is considered as $a_1 \vee \ldots \vee a_m \leftarrow true$, where *true* is a constant atom with its intended interpretation. Sometimes, is better to denote a clause 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, $HEAD(P)$ to denote the set of atoms ocurring in heads in P. We use $\models$ to denote the consequence relation for classical first-order logic. 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)* denotes 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]).

Definition 6.1 (Definition of *def* and *sup*)
Let P be a normal program. Let a be an atom in a given signature $\mathcal{L}$ (such that $\mathcal{L}_P \subseteq \mathcal{L}$), by the definition of a in P, we mean the set of clauses: $\{a \leftarrow body \in P\}$, that we denote by $def(a)$. We define

$$sup(a) := \begin{cases} false & \text{if } def(a) = \emptyset \\ body_1 \vee \ldots \vee body_n & otherwise \end{cases}$$

where $def(a) = \{a \leftarrow body_1, \ldots, a \leftarrow body_n\}$

Definition 6.2 (comp(P) (Clark,1978))
For any normal program P, we define comp(P) over a given signature $\mathcal{L}$ (where $\mathcal{L}_P \subseteq \mathcal{L}$) as the classical theory $\{a \leftrightarrow sup(a) : a \in \mathcal{L}\}$[2].

We use an example to illustrate the above definitions. Let P be the program:

$p \vee q \leftarrow \neg r$
$p \leftarrow s, \neg t$

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

$p \leftarrow \neg r, \neg q$
$q \leftarrow \neg r, \neg p$
$p \leftarrow s, \neg t$

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

$p \leftarrow true$
$q \leftarrow true$
$p \leftarrow s$

[2]In the standard definition $\mathcal{L}$ is $\mathcal{L}_P$. Our paper requires this more general definition

MM(Definite(dis-nor(P)))=$\{p, q\}$. Finally, *comp(dis-nor(P))* over $\mathcal{L}_P$ consists on the formulas

$p \leftrightarrow true \vee s$

$q \leftrightarrow true$

$r \leftrightarrow false$

$s \leftrightarrow false$

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.

Definition 6.3 (Semantics (Brass and Dix,97)
A semantics over a given signature $\mathcal{L}$, is a binary relation $\vdash_s$ between logic programs and pure disjunctions which satisfies the following conditions:

1. *If $P \vdash_s \alpha$ and $\alpha \subseteq \alpha'$, then $P \vdash_s \alpha'$.*
2. *If $\mathcal{A} \leftarrow true \in P$ for a disjunction $\mathcal{A}$, then $P \vdash_s \mathcal{A}$.*
3. *If $a \notin HEAD(P)$ for some atom a, then $P \vdash_s \neg a$.*

It is an implicit assumption that every atom that occurs in P, α, α', $\mathcal{A}$, or a must belong to $\mathcal{L}$.
For any program P we define its minimal semantics as:
$SEM_{min}(P, \mathcal{L}) := \{A |$ *A is a pure disjunction that belongs to every semantics over $\mathcal{L}$ of $P\}$*

This means that the minimal semantics of a program is defined only by the rules 1,2 and 3 just defined. For normal programs we defined a semantics as a binary relation between normal programs and literals that satisfies rules 2 and 3 given above, that is, we get rid of rule 1. For rule 2 note that of course $\mathcal{A}$ reduces to an atom.

Again, the key concept of this approach is the idea of a *transformation* rule.

The following transformations are defined in [Brass and Dix, 1997; Brewka and Dix, 1996] and generalize the corresponding definitions for normal programs.

Definition 6.4 (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 6.1 (Transformation)
Let P be the program:

$a \vee b \leftarrow c, \neg c, \neg d$

$a \vee c \leftarrow b$

$c \vee d \leftarrow \neg e$

$b \leftarrow \neg c, \neg d, \neg e$

then $HEAD(P) = \{a, b, c, d\}$, and
$SEM_{min}(P, \mathcal{L}_P) = \emptyset$.
We can apply **RED$^+$** *to get the program P_1:*

$a \vee b \leftarrow c, \neg c, \neg d$

$a \vee c \leftarrow b$

$c \vee d \leftarrow true$

$b \leftarrow \neg c, \neg d, \neg e$

If we apply **RED$^+$** *again, we get program P_2:*

$a \vee b \leftarrow c, \neg c, \neg d$

$a \vee c \leftarrow b$

$c \vee d \leftarrow true$

$b \leftarrow \neg c, \neg d$

$SEM_{min}(P_2, \mathcal{L}_P)$ =
$\{\{c, d\}, \{c, d, a\}, \{c, d, b\}, \{c, d, a, b\}, \{\neg e\}, \ldots,$
$\{\neg a, \neg b, \neg c, \neg d, \neg e\}\}$.

Clearly $\{c, d, a\} \in SEM_{min}(P_2, \mathcal{L}_P)$ means that $c \vee d \vee a$ is a consequence in $SEM_{min}(P_2, \mathcal{L}_P)$. Now, we can apply **SUB** *to get program P_3:*

$a \vee c \leftarrow b$

$c \vee d \leftarrow true$

$b \leftarrow \neg c, \neg d$

We will refer to this example again, that we will start calling our basic example.

Obviously, the just mentioned transformations are among the minimal requirements a *well-behaved* semantics should have (see [Dix, 1995b]). For this reason, every semantics presented in this paper will be invariant under the transformations RED$^+$, RED$^-$ and SUB.

The following transformations are defined in [Brass and Dix, 1997; Brewka and Dix, 1996].

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)* same as for normal programs.

Let $\mathcal{CS}_5$ be the rewriting system which contains, besides the basic transformation rules, the rules GPPE and TAUT. This system is introduced in [Brass and Dix, 1997] and is confluent and terminating as shown in [Brass and Dix, 1998].

Definition 6.5 (D-WFS)
The disjunctive wellfounded semantics D-WFS is defined as the weakest semantics satisfying SUB, RED$^+$, RED$^-$, GPPE and TAUT.

Let us note that although the $\mathcal{CS}_5$ system has the nice property of confluence (and termination), its computational properties are not that efficient. In fact, computing the normal form of a program is exponential (even for normal programs, whereas it is known that the WFS can be computed in quadratic time).

We introduce our proposed semantics D1-WFS and D1-WFS-COMP and give some important results about them. Unless stated otherwise we assume that every program is a disjunctive program.

Definition 6.6 (Dloop)
For a program P, let $unf(P) := \mathcal{L}_P \setminus MM(Definite(dis - nor(P))$. The transformation **Dloop** *reduces a program P to $P_1 := \{\mathcal{A} \leftarrow \mathcal{B}^+, \neg\mathcal{B}^- |\ \mathcal{B}^+ \cap unf(P) = \emptyset\}$. We assume that the given transformation takes place only if $P \neq P_1$.*

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

Definition 6.7 (Dsuc)
Suppose that P is a program that includes a fact $a \leftarrow true$ and a clause $\mathcal{Q} \leftarrow Body$ such that $a \in Body$. Then we replace this clause by the clause $\mathcal{Q} \leftarrow Body \setminus \{a\}$.

Definition 6.8 ($\mathcal{CS}_6$)
Given a relation R, we define R' as follows:

$$aR'b \; iff \begin{cases} aR^n b \text{ if } \exists n | aR^n b, \neg\exists c | aR^{n+1} c \\ a = b \qquad o.w. \end{cases}$$

Given two relations R_1 and R_2, we define $R_2 \diamond R_1$ as:
$R_2 \circ R_1 \cup \{(x,y) | (x,y) \in R_1, \neg\exists (y,z) \in R_2\}$, *where $\circ$ denotes the standard composition of relations.*
Let **REDUCE** *be the binary relation on programs defined by:*
$Dloop' \diamond (Dsuc' \diamond (SUB \cup RED^+ \cup RED^-)') \setminus I$, *where I denotes the identity relation.*
Finally, let $\mathcal{CS}_6$ be the rewriting system based on the basic transformation **REDUCE**.

Theorem 6.1 (Confl. and termination of $\mathcal{CS}_6$)
The system $\mathcal{CS}_6$ is confluent and terminating. It induces a semantics that we call D1-WFS. If we consider only normal programs then its induced semantics corresponds to the well-founded semantics.

Proof: Confluence is immediate since **REDUCE** behaves as a partial function. Moreover, **REDUCES** always deletes something in the program and so this relation is terminating. For normal programs, the system is clearly equivalent to **RED$^+$ + RED$^-$ + S + Loop + SUB** (that is, both systems define the same normal form) which in turns defines the well-founded semantics.

Consider again P from our *basic example* introduced before. 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 $c \vee d \leftarrow true$. So, **REDUCES** (which can be seen as a macro reduction) transforms P (in one step) to P_5. Since **REDUCES** can not be applied again, P_5 is the normal form of the $\mathcal{CS}_6$ system.

For this example it turns out that D-WFS is equivalent to D1-WFS, but this is false in general. However for normal programs both systems are equivalent since they define WFS, but note that the normal forms for $\mathcal{CS}_5$ and $\mathcal{CS}_6$ are not necessarily the same. An advantage of $\mathcal{CS}_6$ over $\mathcal{CS}_5$ (again for normal programs) is that the normal form of $\mathcal{CS}_6$ is polynomial-time computable, while computing the normal form of $\mathcal{CS}_5$ is in general exponential as it is shown in [Brass *et al.*, 1997].

We now define a very strong semantics that includes the power of *comp*.

Definition 6.9 (D1-WFS-COMP)
For every program P, we define $DCOMP(P) := comp(dis - nor(normal_{\mathcal{CS}_6}(P)))$ over L_P. We define D1-WFS-COMP(P) as the set of pure disjunctions that are logical consequences of DCOMP(P).

It is immediate to see that D1-WFS-COMP is more powerful than D1-WFS. Take for instance the program P:

$p \vee q \leftarrow true$
$r \leftarrow \neg p$
$r \leftarrow \neg q$

Then D-WFS(P) = $\{\{p,q\},\{p,q,r\}\}$ =D1-WFS(P), however, D1-WFS-COMP(P) at least derives r. In this case D1-WFS-COMP corresponds to STABLE, but this is not always true. Sometimes STABLE is inconsistent, while D1-WFS-COMP is not. Consider P as:

$d \vee e$
$c \leftarrow c$
$b \leftarrow a$
$a \leftarrow b$
$a \leftarrow \neg b, \neg c$

Note that STABLE is inconsistent while D1-WFS-COMP is not. This is because DCOMP(P) is:

$d \leftrightarrow \neg e$
$e \leftrightarrow \neg d$
$a \leftrightarrow (b \vee \neg b)$
$c \leftrightarrow false$

Due to its construction, we see that D1-WFS-COMP is similar to STABLE. However, STABLE is inconsistent more often than D1-WFS-COMP (at least for normal programs)

Our current research suggests that logic programming can be extended by adding both disjunctions and

partial-orders as defined and studied in detail in [Osorio *et al.*, 1995, and 1998; Osorio and Jayaraman, 1998; Osorio, 1998] arriving to a very powerful declarative programming language. This integration is one of our main lines of research.

7 Conclusion

We introduced the general notion of a *confluent LP-system*. Such a system is based on certain transformation rules and induces a semantics SEM in a natural way. Thanks to confluence and termination, our transformation rules have both a declarative and an operational meaning. From the declarative point of view, they tell us that our semantics is closed under the given transformation rule. From an operational point of view the transformations are computable functions that can be applied to *simplify* the program.

When we arrive to the normalform, then computing its semantics is immediate. Rationality is a desirable property of semantics in logic programming. Confluence plus a simple property that we call partial distribution implies rationality. It is straightforward to see that all (but one) of our semantics for normal programs satisfy partial distribution. Therefore we know that all but one of these semantics are rational.

We showed that most of the well-known semantics for normal logic programs are induced by confluent LP-systems. Moreover, we showed by introducing several new transformation rules that the corresponding LP-systems induce interesting semantics which are polynomial time computable and extend WFS. We also showed how to apply our approach to disjunctive programs.

It is quite surprising that the simple notion of a *confluent LP-system* that we introduced here, is so flexible that it allows us to extend recently defined calculi for WFS by new transformations in such a way, that the new system still is confluent. We have therefore shown that rewriting methods can be successfully applied in the realm of logic programming semantics.

Acknowledgments

This research was supported in part by grants from CONACyT (number 28452A).

References

Brass S., and J. Dix, "A General Approach to Bottom-Up Computation of Disjunctive Semantics", in Dix J., L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 927, Springer, Berlin, 1995, pp 127–155.

Brewka G., and J. Dix, *Knowledge representation with logic programs*, Technical report, Tutorial Notes of the 12th European Conference on Artificial Intelligence (ECAI '96), 1996. Also appeared as Technical Report 15/96, Dept. of CS of the University of Koblenz-Landau. Will appear as Chapter 6 in *Handbook of Philosophical Logic*, 2nd edition (1998), Volume 6, Methodologies.

Brass S., and J. Dix, "Characterizations of the Disjunctive Stable Semantics by Partial Evaluation", *Journal of Logic Programming*, 32(3):207–228, 1997. (Extended abstract appeared in: Characterizations of the Stable Semantics by Partial Evaluation *LPNMR, Proceedings of the Third International Conference, Kentucky*, pages 85–98, 1995. LNCS 928, Springer.).

Brass S., and J. Dix, "Characterizations of the Disjunctive Well-founded Semantics: Confluent Calculi and Iterated GCWA", *Journal of Automated Reasoning*, to appear, 1998. (Extended abstract appeared in: Characterizing D-WFS: Confluence and Iterated GCWA. *Logics in Artificial Intelligence, JELIA '96*, pages 268–283, 1996. Springer, LNCS 1126.).

Baral C., and M. Gelfond, "Logic Programming and Knowlege Representation", *Journal of Logic Programming*, 19-20, 1994.

Brass S., U. Zukowski, and B. Freitag, "Transformation Based Bottom-Up Computation of the Well-Founded Model", In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 1216, pages 171–201. Springer, Berlin, 1997.

Clark K., "Negation as Failure", In H. Gallaire and J. Minker, editors, *Logic and Data-Bases*, pp. 293–322. Plenum, New York, 1978.

Dix J., "A Framework for Representing and Characterizing Semantics of Logic Programs", In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, pages 591–602. San Mateo, CA, Morgan Kaufmann, 1992.

Dix J., "A Classification-Theory of Semantics of Normal Logic Programs: I. Strong Properties", *Fundamenta Informaticae*, XXII(3):227–255, 1995.

Dix J., "A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties", *Fundamenta Informaticae*, XXII(3):257–288, 1995.

Dix J., and M. Müller, "Partial Evaluation and Relevance for Approximations of the Stable Semantics", in Ras Z.W. and M. Zemankova, editors, *Proceedings of the 8th Int. Symp. on Methodologies for Intelligent Systems, Charlotte, NC, 1994*, LNAI 869, pp. 511–520, Berlin, 1994. Springer.

Dix J., and M. Osorio, *Towards Well-Behaved Semantics Suitable for Aggregation*, Research Report 11-97. University of Koblenz, Germany 1997.

Dix J., J. Arrazola, and M. Osorio, *Confluent*

Rewriting Systems for Logic Programming Semantics, Research Report 27-97. University of Koblenz, Germany 1997.

Dix J., and M. Osorio, "Provability Closures in Logic Programming", in *Proceedings of the International Symposium on Computer Science in Mexico*, to appear, 1997.

Fitting M. C., "A Kripke-Kleene Semantics of logic Programs", *Journal of Logic Programming*, 4:295–312, 1985.

Fitting M. C., "Partial Models and Logic Programming", *Theoretical Computer Science*, 48:229–255, 1986.

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

Gelfond M., and V. Lifschitz, "Logic Program with Classical Negation", in David H.D. Warren and Peter Szeredi, editors, *Proceedings of the 7th Int. Conf. on Logic Programming*, pp. 579–597. MIT, June 1990.

Lloyd J. W., *Foundations of Logic Programming*, 2nd Ed., Springer, Berlin, 1987.

Osorio M., B. Jayaraman, and K. Moon, "Partial Order Programming (revisited)", *Proc. Algebraic Methodology and Software Technology*, pp. 561-575. Springer-Verlag, July 1995.

Osorio M., and B. Jayaraman, "Aggregation and Well-Founded Semantics$^+$," *Proc. 5th Intl. Workshop on Non-Monotonic Extensions of Logic Programming*, pp. 71-90, LNAI 1216, J. Dix, L. Pereira and T. Przymusinski (eds.), Springer-Verlag, 1997.

Osorio M., B. Jayaraman, and D. Plaisted, "Theory of partial-order programming," accepted for publication in *Science of computer programming Journal*, Elsevier.

Osorio M., "Semantics of Partial order programming", presented at JELIA98, Germany, Oct. 1998 appears in *Proc. JELIA98*, pp. 47-61, LNAI 1489, Springer-Verlag, 1998.

Osorio M., and B. Jayaraman, "Integrating the Completion and the Well-Founded Semantics", presented at *Iberamia98*, Portugal, Oct. 1998, accepted for publication in Helder Coelho, editor, *Proc. Iberamia98*, pp. 230-241, LNAI 1484, Springer-Verlag 1998.

Schlipf J. S., "Formalizing a Logic for Logic Programming", *Annals of Mathematics and Artificial Intelligence*, 5:279–302, 1992.

van Gelder A., K. A. Ross, and J. S. Schlipf, "Unfounded Sets and well-founded Semantics for general logic Programs", in *Proceedings 7th Symposion on Principles of Database Systems*, pages 221–230, 1988.

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

Zepeda C., *Semanticas Bien Fundamentadas*, Tesis de maestria, UDLA, Cholula, 1997.

José Ramón Arrazola *Ramírez obtained his BSC., MC., and PhD. degrees in Mathematical Physics from the Universidad Autónoma de Puebla. His PhD. Dissertation was about Lipschitz Functions Algebra. He is currently a professor at the Mathematical Physics Faculty of the same University. He is author and co-author of several publications in Journals and Proceedings, and is member of the National Researchers Systems as a candidate.*

Jürgen Dix is *an Assistant Professor in the Department of Computer Science at University of Koblenz-Landau in Koblenz. He got his PhD in 1992 at Karlsruhe university and his habilitation at TU Vienna in 1996. He is co-editor of 6 proceedings in the Springer* Lecture Notes of Artificial Intelligence *series, co-author of a monograph on* Nonmonotonic Reasoning, *and author of more than 30 articles in Handbooks and Journals.*
He is currently a visiting assistant professor at University of Maryland, College Park and working on Multi-Agents (he is co-authoring a forthcoming book at MIT-Press on «Heterogenous Active Agents»), Data Mining and Uncertainty in Databases.

Mauricio Osorio *got a BSC degree on Computer Science from UAP, a MC degree on Artificial Intelligence from CINVESTAV, and a PhD degree on Computer Science rom the University of New York at Buffalo. Currently, he is an Associate Professor at the Universidad de las Américas, and the coordinator of the Applied Mathematics and Computer Science Laboratory of the Automatics and Information Technologies esearch Center. He has several CONACyT financed projects, and is a member of the National Researchers Systems.*