Measuring Instability in Normal Residuated Logic Programs: Discarding Information

Nicolás Madrid Manuel Ojeda-Aciego

Dept. Matemática Aplicada[⋆]
Univ. Málaga, Spain

Abstract. Inconsistency in the framework of general residuated logic programs can be, somehow, decomposed in two notions: incoherence and instability. In this work, we focus on the measure of instability of normal residuated programs. Some measures are provided and initial results are obtained in terms of the amount of information that have to be discarded in order to recover stability.

1 Introduction

In many fields of automated information processing it becomes crucial to consider together imprecise, uncertain or inconsistent information. Although inconsistency is an undesirable property, it arises naturally in many real-world problems (for instance, consider the integration of information coming from different sources). Anyway, the analysis of inconsistent knowledge-bases can lead us to obtain useful information: for instance, a big number of contradictions in the statements of a suspect of a crime with respect to the forensic evidences may lead us to increase our confidence on his/her being the culprit; a sensor which send data which contradict other sensors may indicate a possible malfunction. In both cases, a good estimation of the degree of inconsistency of the data can help us to estimate the truth-degree up to which this new information can be safely considered.

There are several papers dealing with inconsistency in a classical logic programming framework. For instance, [1] uses consistency restoring rules as a means to recover whenever possible the consistency of a normal logic program; this approach has been used in [13] to formalize negotiations dealing with incomplete information, preferences, and changing goals. The Answer Set Programming (ASP) framework has been used to detect inconsistencies in large biological networks [2]. Argumentation theory is a suitable framework for inconsistency to arise. There are several non-classical approaches to ASP argumentation, some based on possibility theory, some other based on, for instance, fuzzy set theory [7, 12].

The problem of measuring the degree of inconsistency contained in a knowledgebase has been already considered in the literature [4–6]. This approach shows

[⋆] Partially supported by the Spanish Ministry of Science projects TIN06-15455-C03-01 and TIN09-14562-C05-01 & Junta de Andalucía projects FQM-2049 and FQM-5233.

that measuring the inconsistency of a knowledgebase is useful to allow for the comparison of the inconsistency of various knowledgebases. On the other hand, Lozinskii provided a method [8] for defining the quantity of information of a knowledgebase in propositional logic. However, that method is not suitable when the knowledgebase is inconsistent. Furthermore, it is certainly false that all inconsistent knowledgebases contain the same (null) amount of information, this is especially relevant when considering fuzzy extensions of the theory.

This work is based on the Fuzzy Answer Set Programming for residuated logic programs defined in [9,10], in which we consider a fuzzy answer set attending to two dimensions: coherence and stability, the former is related to strong negation, whereas the latter is related to default negation and the GL-reduct [3]. An inconsistent fuzzy program is a program without fuzzy answer sets, and this can be due to the lack of stable models (instability) or, perhaps, to the inconsistency of every stable model (incoherence). This is why we talk about the two dimensions of inconsistency. In [11] some measures of inconsistency were defined in terms of incoherence; in this work, we aim at providing an initial step towards the measuring the degree of instability in normal residuated logic programs.

The structure of the paper is described as follows. In Section 2 we recall the definition of stable model. Section 3 describes the possible causes of the instability of a residuated logic program and defines the notion of information measure, which assigns a degree of information to any value in the truth space. In Section 4 we define the measure of instability which establish how many information has to be deleted from a set of rules in order to recovering the stability in the residuated logic program.

2 Preliminaries

Let us start this section recalling the definition of residuated lattice, which fixes the set of truth values and the relationship between the conjunction and the implication (the adjoint condition) occurring in our logic programs.

Definition 1. *A* residuated lattice *is a tuple* $(L, \leq, *, \leftarrow)$ *such that:*

1. $(L, \leq)$ *is a complete bounded lattice, with top and bottom elements 1 and 0.*
2. $(L, *, 1)$ *is a commutative monoid with unit element 1.*
3. $(*, \leftarrow)$ *forms an adjoint pair, i.e.* $z \leq (x \leftarrow y)$ *iff* $y * z \leq x \quad \forall x, y, z \in L$.

In the rest of the paper we will consider a residuated lattice enriched with a negation operator, $(L, *, \leftarrow, \neg)$. The negation $\neg$ will model the notion of default negation often used in logic programming. As usual, a negation operator, over L, is any decreasing mapping $n \colon L \to L$ satisfying $n(0) = 1$ and $n(1) = 0$. In the examples, we will use the following familiy of negation operators:

$$n_\alpha(x) = \begin{cases} 1 & \text{if } x \leq \alpha \\ 0 & \text{if } x > \alpha \end{cases} \qquad\qquad n(x) = 1 - x$$

Definition 2. *Given a residuated lattice with negation* $(L, \leq, *, \leftarrow, \neg)$*, a* normal residuated logic program $\mathbb{P}$ *is a set of weighted rules of the form*

$$\langle p \leftarrow p_1 * \cdots * p_m * \neg p_{m+1} * \cdots * \neg p_n; \quad \vartheta \rangle$$

where ϑ *is an element of* L *and* $p, p_1, \ldots, p_n$ *are propositional symbols.*

It is usual to denote the rules as $\langle p \leftarrow \mathcal{B}; \vartheta \rangle$. The formula $\mathcal{B}$ is usually called the *body* of the rule whereas p is called its *head.* A *fact* is a rule with empty body, i.e facts are rules with this form $\langle p \leftarrow \quad ; \vartheta \rangle$. The set of propositional symbols appearing in $\mathbb{P}$ is denoted by $\Pi_{\mathbb{P}}$.

Definition 3. *A fuzzy* L-interpretation *is a mapping* $I \colon \Pi_{\mathbb{P}} \to L$*; note that the domain of the interpretation can be lifted to any rule by homomorphic extension.*

We say that I satisfies *a rule* $\langle \ell \leftarrow \mathcal{B}; \quad \vartheta \rangle$ *if and only if* $I(\mathcal{B}) * \vartheta \leq I(\ell)$ *or, equivalently,* $\vartheta \leq I(\ell \leftarrow \mathcal{B})$.

Finally, I *is a* model *of* $\mathbb{P}$ *if it satisfies all rules (and facts) in* $\mathbb{P}$.

Note that the order relation in the residuated lattice $(L, \leq)$ can be extended over the set of all L-interpretations as follows: *Let* I *and* J *be two* L*-interpretations, then* $I \leq J$ *if and only if* $I(p) \leq J(p)$ *for all propositional symbol* $p \in \Pi_{\mathbb{P}}$.

Stable Models

Our aim in this section is to adapt the approach given in [3] to the normal residuated logic programs just defined in the section above.

Let us consider a normal residuated logic program $\mathbb{P}$ together with a fuzzy L-interpretation I. To begin with, we will construct a new normal program $\mathbb{P}_I$ by substituting each rule in $\mathbb{P}$ such as

$$\langle p \leftarrow p_1 * \cdots * p_m * \neg p_{m+1} * \cdots * \neg p_n; \quad \vartheta \rangle$$

by the rule[1]

$$\langle p \leftarrow p_1 * \cdots * p_m; \quad \neg I(p_{m+1}) * \cdots * \neg I(p_n) * \vartheta \rangle$$

Notice that the new program $\mathbb{P}_I$ is positive , that is, does not contain any negation; in fact, the construction closely resembles that of a reduct in the classical case, this is why we introduce the following:

Definition 4. *The program* $\mathbb{P}_I$ *is called the* reduct *of* $\mathbb{P}$ *wrt the interpretation* I.

[1] Note the overloaded use of the negation symbol, as a syntactic function in the formulas and as the algebraic negation in the truth-values.

As a result of the definition, note that given two fuzzy L-interpretations I and J, then the reducts $\mathbb{P}_I$ and $\mathbb{P}_J$ have the same rules, and might only differ in the values of the weights. By the monotonicity properties of $*$ and $\neg$, we have that if $I \leq J$ then the weight of a rule in $\mathbb{P}_I$ is greater or equal than its weight in $\mathbb{P}_J$.

It is not difficult to prove that every model M of the program $\mathbb{P}$ is a model of the reduct $\mathbb{P}_M$.

Recall tha*t a fuzzy interpreta*tion can be interpreted as a L-fuzzy subset. Now, as usual, the notion of reduct allows for defining a *stable set* for a program.

Definition 5. *Let $\mathbb{P}$ be a normal residuated logic program and let I be a fuzzy L-interpretation; I is said to be a* stable set *of $\mathbb{P}$ iff I is the least model of $\mathbb{P}_I$.*

Theorem 1. *Any stable set of $\mathbb{P}$ is a minimal model of $\mathbb{P}$.*

Thanks to Theorem 1 we know that every stable set is a model, therefore we will be able to use the term stable model to refer to a stable set. Obviously, this approach is a conservative extension of the classical approach. Note, as well, that a residuated logic program can have infinitely many stable models.

In the following example we use a simple normal logic program with just one rule in order to clarify the definition of stable set (stable model).

Example 1. Consider the program $\langle p \leftarrow \neg q \quad ; \vartheta \rangle$. Given a fuzzy L-interpretation $I \colon \mathit{\Pi} \to L$, the reduct $\mathbb{P}_I$ is the rule (actually, the fact) $\langle p \quad ; \vartheta * \neg I(q) \rangle$ for which the least model is $M(p) = \vartheta * \neg I(q)$, and $M(q) = 0$. As a result, I is a stable model of $\mathbb{P}$ if and only if $I(p) = \vartheta * \neg I(q) = \vartheta * \neg(0) = \vartheta * 1 = \vartheta$ and $I(q) = 0$. □

The following example shows that stable models for a normal residuated logic program need not exist.

Example 2. Consider the the following normal residuated logic program on the product logic

$$\langle p \leftarrow \neg p \quad ; 1 \rangle$$

defined over the residuated lattice $([0,1], \leq, *_P, \leftarrow_P, n_\alpha)$ (for any $\alpha \in [0,1)$). This normal residuated logic program does not have stable models. Let I be an interpretation. The reduct w.r.t. I is either the fact $\langle p \leftarrow \quad ; 1 \rangle$ if $I(p) \leq \alpha$ or the fact $\langle p \leftarrow \quad ; 0 \rangle$ if $I(p) > \alpha$. In any case, if I is a stable model then $I(p)$ is equal either 1 or 0. However, none of the interpretations is stable model of this normal residuated logic program. □

The aim of this work is to study normal residuated logic programs without any stable model by means of measures which determine how much information one has to add or delete in order to recover at least one stable model. We start by proposing the following definition:

Definition 6. *A normal residuated logic program $\mathbb{P}$ is stable if and only if there is an L-interpretation I that is a stable model of $\mathbb{P}$; i.e I is the least model of $\mathbb{P}_I$. Otherwise, $\mathbb{P}$ is called unstable.*

3 Causes of instability: measures of information

Instability is an undesirable feature of a logic program. When representing knowledge as a (residuated) logic program it is usual to implement rules according to a set of external data (obtained either from sensors or from suggestion of an expert); this data is subject to mistake and/or imprecisions, and may lead to the following shortcomings:

– Not to include relevant information. (Missing information)
– Include information which is either false or leading to contradiction. (Excess of information)

Any of the situations above might lead to instability. Let us further discuss this by means of an example: the following program tries to simulate a procedure to deduce which sports are practised by a person give some data.

$$\begin{aligned}
r_1\colon &\langle \mathit{Football} \leftarrow n_{0.4}(\mathit{Basketball}) *_G \mathit{LivesInSuburb} *_G \mathit{AthleticBody} \quad ; 0.6\rangle\\
r_2\colon &\langle \mathit{Basketball} \leftarrow n_{0.4}(\mathit{Cycling}) *_G \mathit{Tall} *_G \mathit{AthleticBody} \quad ; 0.6\rangle\\
r_3\colon &\langle \mathit{Cycling} \leftarrow n_{0.4}(\mathit{Football}) *_G \mathit{Slim} *_G \mathit{AthleticBody} \quad ; 0.6\rangle
\end{aligned}$$

The first rule determines that if a person with an athletic body, which lives in a suburb and we do not know whether he practices regularly basketball, then this person practices football frequently (the interpretation of the other two rules is similar). These three rules do not imply any contradiction, in fact, the program consisting of the three rules has just one stable model[2] $I_\perp$. However, if we add the following facts

$$\begin{array}{ll}
r_4\colon \langle \mathit{AthleticBody} \leftarrow \quad ; 0.8\rangle & r_5\colon \langle \mathit{LivesInSuburb} \leftarrow \quad ; 1\rangle\\
r_6\colon \langle \mathit{Tall} \leftarrow \quad ; 0.7\rangle & r_7\colon \langle \mathit{Slim} \leftarrow \quad ; 0.8\rangle
\end{array}$$

the program turns out to be unstable. What are the reasons for this behaviour?

As we said above, it may be because of excess or lack of information. For the former, excess of information can reside in any of the seven rules, it might be that too much information is obtained by default from r_1, r_2 and r_3. Notice that if the weights are changed to 0.39, therefore reducing the amount of information provided by those rules, the program would remain stable. Lack of information is more difficult to handle, in that it is not possible to know which rules are needed; it might be just a fact (if we include the fact $\langle \mathit{Football} \leftarrow \quad ; 0.5\rangle$, the program gets stable again), or a more complex rule or set of rules.

In this work we focus on the treatment of excess of information, and we propose a framework to measure the instability of a program by means of the minimum amount of information which we have to delete in order to obtain a stable program. Our approach to reducing the amount of information provided by a program is based of the values of the weights, since the smaller they are the less information is produced. The key point is how to measure the amount of information which is eliminated.

We propose to fix an operator $m\colon L \to \mathbb{R}^+$ such that:

[2] $I_\perp$ denotes the bottom element of the complete lattice of L-interpretations.

– $m(x) = 0$ if and only if $x = \bot$
– m is monotonic

Such an operator will be called an *information measure.*

It is not difficult to provide examples of these operators in the unit interval or in any finite lattice:

Example 3. Any norm $||\cdot||$ on the lattice $([0,1], \leq)$ is an information measure, since $||x|| = 0$ if and only if $x = 0$; and if $x \leq y$ then

$$||x|| = ||\frac{x}{y} \cdot y|| = |\frac{x}{y}| \cdot ||y|| \leq ||y||$$

□

Example 4. Let $(L, \leq)$ be a finite lattice. An information measure can be defined as follows:

$$m(x) = \max\{n \colon \bot < x_1 < \cdots < x_n = x\}$$

Let us check that, in fact, it is an information measure: if $x \neq \bot$, then $\bot < x$, and this implies $m(x) \geq 1$. On the other hand, if $x < y$, then for all chain $\bot < x_1 < \cdots < x_n = x$ we have the chain $\bot < x_1 < \cdots < x_n = x < x_{n+1} = y$ which has a greater length, and this implies $m(x) < m(y)$. □

Information measures will be used to determine the amount of information inherently contained in any element of the lattice. From now on, we will consider that any lattice has an associated information measure.

4 Measuring instability of normal residuated programs

In this section we define an instability measure based on the amount of information deleted from a unsatble program so that it gets stable. Contrariwise to the classical case, in which the only form to delete information is by deleting rules completely, in our framework we can just reduce their weights by some amount. A specific operator will be defined for this task.

For that purpose, we need to fix a t-norm t to handle the values of $\mathcal{L}$ (recall that a t-norm is a commutative and monotonic map $L \times L \to L$ satisfying $t(\bot, x) = \bot$ and $t(\top, x) = x$). Fixed such a t-norm, we can define an operator to modify the weights of rules.

Given a normal residuated logic program $\mathbb{P}$, a set $\{\langle r_i; \vartheta_i\rangle\}_i$ of rules in $\mathbb{P}$ and a set of values $\{\varphi_i\}_i$ we define a new general residuated logic program $O_{\mathbb{P}}(\{\langle r_i; \vartheta_i\rangle\}_i, \{\varphi\}_i)$ as follows:

$$O_{\mathbb{P}}(\{\langle r_i; \vartheta_i\rangle\}_i, \{\varphi\}_i) = (\mathbb{P} \smallsetminus \{\langle r_i; \vartheta_i\rangle\}_i) \cup \{\langle r_i; t(\vartheta_i, \varphi_i)\rangle\}_i$$

In other words, the operator $O_{\mathbb{P}}$ substitutes the weight of any rule $\langle r_j; \vartheta_j\rangle$ in the given set by $t(\vartheta_j, \varphi_j)$.

It is not difficult to note that the resulting program has smaller weights than the original one. The following example illustrates this fact.

Example 5. Consider the residuated lattice with negation $([0,1], \leq, *_P, \leftarrow_P, n)$, and the following residuated program

$$\begin{array}{rlrl} r_1\colon & \langle p \leftarrow q * t * \neg t \quad ; 0.7\rangle & r_2\colon & \langle p \leftarrow t * \neg s \quad ; 0.8\rangle \\ r_3\colon & \langle q \leftarrow \neg v \quad ; 0.2\rangle & r_4\colon & \langle t \leftarrow s * u * \neg v \quad ; 0.9\rangle \end{array}$$

Assume the product t-norm $(t(x,y) = x \cdot y)$ as the t-norm associated to the operator $O_{\mathbb{P}}$. Then, the program $O_{\mathbb{P}}(\{r_1, r_4\}, \{0.5, 0.9\})$ is shown below:

$$\begin{array}{rlrl} r_1\colon & \langle p \leftarrow q * t * \neg t \quad ; 0.35\rangle & r_2\colon & \langle p \leftarrow t * \neg s \quad ; 0.8\rangle \\ r_3\colon & \langle q \leftarrow \neg v \quad ; 0.2\rangle & r_4\colon & \langle t \leftarrow s * u * v \quad ; 0.81\rangle \end{array}$$

Notice that the weights of rules r_1 and r_4 are reduced by a factor 0.5 and 0.9 respectively. $\Box$

The instability measure will be defined in terms of the amount of discarded information needed to get stability, and this will be computed by means of an information measure, as those introduced in Section 3, and the formula

$$\sum_{i \in \mathbb{I}} \big(m(\top) - m(\varphi_i) \big)$$

The sum above, in some sense, measures the amount of information discarded from the program; the lesser the values of φ_i the more information discarded, and greater the sum. Notice as well that $O_{\mathbb{P}}$ does not reduce the weights of the program if and only if φ_i are $\top$ for all i, and the previous sum reduces to 0.

Example 6. Continuing with Example 5, if we consider in $[0,1]$ the information measure induced by the Euclidean norm, then the amount of discarded information by the use of $O_{\mathbb{P}}(\{r_1, r_4\}, \{0.5, 0.9\})$ would be $(1 - 0.5) + (1 - 0.9) = 0.6$. $\Box$

Now, we can define the following instability measure, given a general residuated logic program $\mathbb{P}$ and a set of rules $\{\langle r_i, \vartheta_i\rangle\}_i \subseteq \mathbb{P}$ (w.r.t. the respective residuated logic program) as:

$$\textsc{Instab}_{\mathbb{P}}(\{\langle r_i, \vartheta_i\rangle\}_i\}) = \inf\{\sum_{i \in \mathbb{I}} m(\top) - m(\varphi_i) \colon O_{\mathbb{P}}(\{\langle r_i, \vartheta_i\rangle\}_i, \{\varphi\}_i) \textit{ is stable } \}$$

It is important to note that this operator needs not be defined for any set of rules (the sum could be infinite). This is not a big problem, as that would indicate that it is not possible to recover stability by not even discarding completely all the rules in the set.

Example 7. On the residuated lattice with negation $([0,1], \leq, \wedge_P, \leftarrow_P, n_{0.4})$, let us consider the following unstable logic program:[3]

$$\begin{array}{rrl} r_1\colon & \langle p \leftarrow s \wedge \neg q & ; 0.8\rangle \\ r_2\colon & \langle q \leftarrow \neg r \wedge \neg u & ; 0.8\rangle \\ r_3\colon & \langle r \leftarrow \neg p & ; 0.5\rangle \\ r_4\colon & \langle s \leftarrow & ; 0.8\rangle \\ r_5\colon & \langle t \leftarrow \neg p \wedge \neg s & ; 0.5\rangle \\ r_6\colon & \langle v \leftarrow u \wedge \neg r & ; 0.7\rangle \end{array}$$

[3] To increase readability, the subscripts P have been removed.

It is not difficult to check that this program does not have stable models. We will use the product t-norm and the Euclidean norm in the formulas above to measure the instability of the rules of the program. For the case of r_1, one can see that if its weight would be a value $\alpha \leq 0.5$, then the program would have a stable model; specifically, $M \equiv \{(p, 0.8 \cdot \alpha); (q, 0); (r, 0.5); (s, 0.8); (t, 0.4); (v, 0)\}$.

On the other hand, it is possible to set the weight of r_1 to 0.5 using the factor $\varphi = 0.625$. Therefore, the least amount of information to be discarded from r_1 has to be $1 - 0.625 = 0.375$. In other words, $\textsc{Instab}_{\mathbb{P}}(\{r_1\}) = 0.375$. Similarly, we can obtain the instability measures for the rest of rules:

x	r_1	r_2	r_3	r_4	r_5	r_6
$\textsc{Instab}_{\mathbb{P}}(\{x\})$	0.375	0.5	0.2	0.375	$\star$	$\star$

The symbol $\star$ for rules r_5 and r_6 denotes that it is impossible to get a stable program by reducing the weights of these rules. Notice that these results state that, in recovering stability by modifying just one rule, we need to discard much more information from r_2 than in r_3. $\square$

A couple of straightforward results about the instability measure $\textsc{Instab}_{\mathbb{P}}$ are presented below. The first one establishes a relationship between stable programs and zero measure.

Proposition 1. *Let $\mathbb{P}$ be a normal residuated logic program:*

- *If $\mathbb{P}$ is stable then* $\textsc{Instab}_{\mathbb{P}}(\mathbb{P}) = 0$
- *If* $\textsc{Instab}_{\mathbb{P}}(\mathbb{P}) = 0$ *then for all $\varepsilon > 0$ there exists a set $\{\varphi_i\} \subseteq L$ such that $O_{\mathbb{P}}(\{\langle r_i; \vartheta_i\rangle\}_i, \{\varphi\}_i)$ is stable and $\sum_{i \in \mathbb{I}} \big(m(\top) - m(\varphi_i)\big) < \varepsilon$.*

The following proposition states the antitonicity of the measure $\textsc{Instab}_{\mathbb{P}}$:

Proposition 2. *Let $\mathbb{P}$ be a normal residuated logic program and let $\{\langle r_i; \vartheta_i\rangle\} \subseteq \{\langle \overline{r_i}; \overline{\vartheta_i}\rangle\}$ be two sets of rules of $\mathbb{P}$. Then:*

$$\textsc{Instab}_{\mathbb{P}}(\{\langle r_i; \vartheta_i\rangle\}) \geq \textsc{Instab}_{\mathbb{P}}(\{\langle \overline{r_i}; \overline{\vartheta_i}\rangle\})$$

Computing $\textsc{Instab}_{\mathbb{P}}(\{\langle r_i; \vartheta_i\rangle\})$

The aim of this section is to show that computing the value of $\textsc{Instab}_{\mathbb{P}}(\{\langle r_i; \vartheta_i\rangle\})$ is equivalent to computing the set of stable models of a specific logic program. To facilitate the presentation, let us assume that $[0, 1]$ is the set of truth values and the set of rules $\{\langle r_i; \vartheta_i\rangle\}$ is a singleton.

To compute the measure of instability $\textsc{Instab}_{\mathbb{P}}$ we have to obtain what values $\lambda \in [0, 1]$ satisfy that $O_{\mathbb{P}}(\langle r_i; \vartheta_i\rangle, \lambda)$ is stable; we recall that $O_{\mathbb{P}}(\langle r_i; \vartheta_i\rangle, \lambda)$ coincides with $\mathbb{P}$ except in the rule $\langle r_i; \vartheta_i\rangle$, which is changed by $\langle r_i; t(\vartheta_i, \lambda)\rangle$. How can we introduce the parameter λ in $\mathbb{P}$ through propositional symbols? Let α and β be two propositional symbols not occurring in $\mathbb{P}$. Consider the following set of rules:

$$\langle \alpha \leftarrow \neg\beta \quad ; 1\rangle \tag{1}$$

$$\langle \beta \leftarrow \neg\alpha \quad ; 1\rangle \tag{2}$$

where the negation is the standard one $n(x) = 1 - x$. The set of stable models of this pair of rules is the set $\{M_\lambda \equiv (\alpha, \lambda); (\beta, 1-\lambda)\}_{\lambda \in [0,1]}$. Notice that for any $\lambda \in [0, 1]$ there exists a stable model such that $M_\lambda(\alpha) = \lambda$. We consider now a new residuated logic program $\mathbb{P}^\star$ by modifying r_i as follows[4]

$$r_i^\star \colon \langle p_i \leftarrow \mathcal{B} * t(\vartheta, \alpha) \quad ; 1\rangle$$

and including the rules (1) and (2). Then the following proposition holds:

Proposition 3. *Let $\mathbb{P}$ be a residuated logic program. M is a stable model of $\mathbb{P}^\star$ if and only if $M|_{\Pi_\mathbb{P}}$ is a stable model of $O_\mathbb{P}(\langle r_i; \vartheta_i\rangle, M(\alpha))$.*

The above proposition shows that there is a univocal correspondence among the stable model of $\mathbb{P}^\star$ and the parameters λ_i such that $O_\mathbb{P}(\langle r_i; \vartheta_i\rangle, \lambda)$ is stable. Therefore we can compute $\textsc{Instab}_\mathbb{P}(\{\langle r_i; \vartheta_i\rangle\})$ by using the stable models of $\mathbb{P}^\star$:

Corollary 1. *Let $\mathbb{P}$ be a residuated logic program. Then:*

$$\textsc{Instab}_\mathbb{P}(\langle r_i; \vartheta_i\rangle) = \inf\{m(\top) - m(M(\alpha)) \colon M \textit{ is a stable model of } \mathbb{P}^\star\}$$

5 Conclusions

We have continued our study of fuzzy answer set semantics for residuated logic programs by focusing on the measure of instability of normal residuated programs. Some measures have been provided and initial results have been obtained, in terms of the amount of information that has to be discarded in order to recover stability.

As future work, we will study the dual situation in which stability can be recovered by adding information (as in the framework of consistency restoring rules). In addition, we will extend this methodology to provide explanations for inconsistencies in the data by determining minimal representations of conflicts. In practice, this can be used to identify unreliable data or to indicate missing reactions.

References

1. M. Balduccini and M. Gelfond. Logic programs with consistency-restoring rules. In *Intl Symp on Logical Formalization of Commonsense Reasoning, AAAI 2003*, pages 9–18, 2003.

[4] Technically, the resulting rule is not residuated, but the semantics can be easily adapted to cope with this change.

2. M. Gebser, T. Schaub, S. Thiele, B. Usadel, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. In *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 130–144, 2008.
3. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP-88*, pages 1070–1080, 1988.
4. J. Grant and A. Hunter. Measuring inconsistency in knowledgebases. *J. Intell. Inf. Syst.*, 27(2):159–184, 2006.
5. A. Hunter and S. Konieczny. Approaches to measuring inconsistent information. In *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*, pages 191–236, 2005.
6. A. Hunter and S. Konieczny. Measuring inconsistency through minimal inconsistent sets. In *Proc of Principles of Knowledge Representation and Reasoning (KR'08)*, pages 358–366. AAAI Press, 2008.
7. J. Janssen, M. De Cock, and D. Vermeir. Fuzzy argumentation frameworks. In *Proc. of IPMU'08*, pages 513–520, 2008.
8. E. Lozinskii. Information and evidence in logic systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:163—193, 1994.
9. N. Madrid and M. Ojeda-Aciego. Towards a fuzzy answer set semantics for residuated logic programs. In *Proc of WI-IAT'08. Workshop on Fuzzy Logic in the Web*, pages 260–264, 2008.
10. N. Madrid and M. Ojeda-Aciego. On coherence and consistence in fuzzy answer set semantics for residuated logic programs. *Lect. Notes in Computer Science*, 5571:60–67, 2009.
11. N. Madrid and M. Ojeda-Aciego. On the measure of incoherence in extended residuated logic programs. In *IEEE Intl Conf on Fuzzy Systems (FUZZ-IEEE'09)*, pages 598–603, 2009.
12. J. C. Nieves, U. Cortés, and M. Osorio. Possibilistic-based argumentation: An answer set programming approach. *Mexican International Conference on Computer Science*, pages 249–260, 2008.
13. T. C. Son and C. Sakama. Negotiation using logic programming with consistency restoring rules. In *IJCAI'09: Proc 21st Intl Joint Conf on Artificial intelligence*, pages 930–935. Morgan Kaufmann Publishers Inc., 2009.