Towards a Logic-programming System to Debug ASP Knowledge Bases

J. C. Acosta-Guadarrama[*]

Computer Systems
Autonomous University of Mexico State,
FI-UAEM, Mexico
jguadarrama@gmail.com

Abstract. This paper is a characterisation in terms of Weak Constraints both for Minimal Generalised Answer Sets and Optimal Answer Sets, of an updates semantics that provides a solid foundation for the implementation of a system to debug knowledge bases. The proposed system can be employed both to identify conflicts with upcoming information from a dynamic changing environment, and to locate the source of conflict from a given inherent inconsistent knowledge base that is static. In addition, this work introduces some formal specifications towards the implementation of a debugging solver with no need to use a total-order semantics. Accordingly, the two-fold contribution of this work is to show a simpler theoretical framework of preferences characterised in DLV, as well as general foundation to debug both static and dynamic logic programs.

1 Introduction

As one of the major and traditional topics in Artificial Intelligence over the last years, Knowledge Representation and Reasoning has proved to be a discipline that provides strong theoretical methods to manage knowledge bases. In particular, this topic has become more widely applied in administration of knowledge in Ambient Intelligence and of intelligent (rational) agents, and it is especially useful in situations of incomplete knowledge from a *changing environment.* This paper is a proposal to debug knowledge bases that change over time, which is an important topic both in static and dynamic knowledge, and has to be solved before other more-general problems of multiple agents. The proposed framework shall be useful both to find conflicts when new contradictory information arises, and to locate conflicting information from a given static knowledge base, with contradictory information.

There are several earlier related works that have addressed the problem of debugging knowledge bases. For instance, Balduccini and Gelfond [3] proposed a framework to restore consistency of a given list of planning specifications encoded into an ASP static[1] program [3]. The proposal introduces the use of a method

[*] This project is supported by The Mexican Council of Science and Technology, CONACYT.

[1] It is static in the sense that the thorough knowledge base endures no change, while the dynamic part is the generation of plans itself.

called *Consistency-Restoring Rules* to diagnose the source of conflict in a very particular context. Another proposal is when changing "factory" specifications of the knowledge base of an intelligent agent coping with a dynamic changing environment, by means of program transformations, proposed by Acosta G. et al. [2]. Although the latter is in the same context we suggest, they both have not been extended to the general case of updating knowledge bases of an intelligent agent, besides having to move to the particular knowledge base syntax they propose.

As a result, the present work introduces a general alternative to those proposals, and it addresses both kinds of problems into a single framework, and without need of an expanded language. This paper introduces a prototype for updates of knowledge bases, into a more concise and precise method of updates. In addition to that, it introduces some formal specifications for its implementation on top of an ASP solver called DLV [7], without need to use a total-order semantics.

To begin with, one of the latest proposals aiming to meet most well-known principles for changing knowledge bases comes from [9, 1] who, similarly to Balduccini and Gelfond [3], put forward an approach based upon an abductive programming semantics called *Generalised Answer Sets* [6]—or GAS hereafter. The formulation satisfies several well-accepted properties, overcoming known problems from others by taking advantage of its flexibility and simplicity for its ASP foundation.

In need of a solver for the semantics of Acosta G. [1], this twofold contribution comprises both a characterisation of this update semantics in terms of Weak Constraints to debug knowledge bases, as well as a general foundation to implement it. They both contrast with other works in the literature, which implement a semantics for update sequences, rather than successive updates, besides the lack of a formal characterisation in Weak Constraints. Such a lack is also Balduccini and Gelfond's [2003] observation of this domain.

In general, this paper is organised as follows. It starts with preliminary notation and foundation in Section 2, ending with the main definitions of the implemented semantics (Section 2.4). The core of the paper is Section 3, divided into preferred models and debugging knowledge bases. The paper ends with a section of some concluding remarks on the proposal.

2 Preliminaries

In this section we expect the reader to be familiar with basic notions of logic programming in ASP and non-monotonic reasoning from the literature. Before going straight to the system description, though, a short background is still necessary.

2.1 Logic Programming and Answer Sets

The following gives the description of ASP, which is identified with other names like *Stable Logic Programming* or *Stable Model Semantics* [5] and A-Prolog. Its formal language and some more notation are introduced from the literature as follows.

Definition 1 (ASP Language of logic programs, $\mathcal{L}_{\mathsf{ASP}}$). *In the following $\mathcal{L}_{ASP}$ is a language of propositional logic with propositional symbols: $a_0, a_1, \ldots$; connectives: "," (conjunction) and meta-connective ";"; disjunction, denoted as "|"; $\leftarrow$ (derivation, also denoted as $\rightarrow$); propositional constants $\bot$ (falsum); $\top$ (verum); "$\neg$" (default negation or weak negation, also denoted with the* not *word); "$\sim$" (strong negation, equally denoted as "$-$"); auxiliary symbols: "(", ")" (parentheses). The propositional symbols are also called atoms or atomic propositions. A* literal *is an atom or a strong-negated atom. A* rule *ρ is an ordered pair $\mathsf{Head}(\rho) \leftarrow \mathsf{Body}(\rho)$.*

With the notation introduced in Definition 1, one may construct clauses of the following general form that are well known in the literature.

Definition 2 (EDLP). *An* extended disjunctive logic program *is a set of rules of form*

$$\ell_1 \vee \ell_2 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m, \neg \ell_{m+1}, \ldots, \neg \ell_n \tag{1}$$

where ℓ_i is a literal and $0 \leq l \leq m \leq n$.

Although ASP is our main basis, a more flexible means is necessary to set up preferences amongst models, so that one may choose the most appropriate, according to general principles and postulates. One of such intermediate mechanisms is *Abductive Logic Programming*, due to Kakas and Mancarella, briefly presented in the following.

2.2 Minimal Generalised Answer Sets

Balduccini and Gelfond [3] were the first to employ the concept of Minimal Generalised Answer Sets (MGAS) to interpret abductive programs, which provides a more general and flexible framework than standard ASP. The following set of definitions illustrate such a concept.

Definition 3 (Kakas and Mancarella [6]). *An* abductive logic program *is a pair $\langle \Psi, \mathcal{A}^* \rangle$ where Ψ is an arbitrary program and $\mathcal{A}^*$ a set of literals, called abducibles.*

On the other hand, there already exists a semantics to interpret abductive programs, called *generalised answer sets* (GAS) due to Kakas and Mancarella.

Definition 4 (GAS, Kakas and Mancarella [6]). *The expression $\mathcal{M}(\Delta)$ is a* generalised answer set *of the abductive program $\langle \Psi, \mathcal{A}^* \rangle$ if and only if $\Delta \subseteq \mathcal{A}^*$ and $\mathcal{M}(\Delta)$ is an answer set of $\Psi \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$.*

In case there are more than one generalised answer sets, an inclusion order may be established:

Definition 5 (Balduccini and Gelfond [3]). *Let $\mathcal{M}(\Delta_1)$ and $\mathcal{M}(\Delta_2)$ be generalised answer sets of $\langle \Psi, \mathcal{A}^* \rangle$. The relation $\mathcal{M}(\Delta_1) \leq_{\mathcal{A}^*} \mathcal{M}(\Delta_2)$ holds if and only if $\Delta_1 \subseteq \Delta_2$.*

Last, one can easily establish the minimal generalised answer sets from an abductive inclusion order with the following definition.

Definition 6 (MGAS, Balduccini and Gelfond [3]). *Let $\mathcal{M}(\Delta)$ be a* minimal generalised answer set *(MGAS) of $\langle \Psi, \mathcal{A}^* \rangle$ if and only if $\mathcal{M}(\Delta)$ is a generalised answer set of $\langle \Psi, \mathcal{A}^* \rangle$ and it is minimal with respect to its* abductive inclusion order*.*

2.3 Weak Constraints

Leone et al. [7] have introduced a nice feature of DLV solver known as *Weak Constraints* that may be employed to set up preferences between models. In particular, a weak constraint is a variant of an *integrity constraint* that may be violated in order to establish priorities amongst models. One of its differences is the introduction of a new derivation symbol ":∼", rather than ":– " or "←". Moreover, one can specify priority levels and weights of constraints to minimise, in the end, the sum of weights (of the violated weak constraints) at the highest priority levels. Formally,

Definition 7 (Weak-constraint Expression, [7]). *A* weak-constraint *(ω) is an expression of the form ($:\sim \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m [w : p]$), where for $0 \leq k \leq m$, $\ell_1, \ldots, \ell_m$ are literals, while w (the weight) and p (the level, or layer) are positive integer constants or variables.*

In addition, $\Omega(\Psi)$ shall denote the finite *set of weak constraints* occurring in a given program Ψ. Likewise, a *ω-program* is an EDLP with weak constraints.

In order to provide a more syntactic sugar and a more ASP flavour, another way to define a weak-consitraint expression from Definition 7 is as follows.

Definition 8 (Weak-constraint Rule). *A* weak-constraint rule *(ω) is an expression of the form*

$$[w : p] \leftarrow \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m \tag{2}$$

where for $0 \leq k \leq m$, $\ell_1, \ldots, \ell_m$ are literals, while w (the weight) and p (the level, or layer) are positive integer constants or variables.

From now on, the weak-constraints form in Definition 7 shall be employed in the context of DLV-code, while Definition 8 in other more-general contexts.

Similarly to integrity constraints in Section 2.1, one may say that a weak-constraint rule $\rho = ([w : p] \leftarrow \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m)$ is *violated* by an answer set $\mathcal{S}$ of a program $\Psi \setminus \{\rho\}$ if the following three conditions hold:

1. $\rho \in \Psi$
2. $\{\ell_1, \ldots, \ell_k\} \subseteq \mathcal{S}$

3. $\{\ell_{k+1}, \ldots, \ell_m\} \nsubseteq \mathcal{S}$

Additionally, Leone et al. simplify the combination of weights in levels by introducing a function $H^{\Psi}(\mathcal{S})$ that grows in direct proportion to the weight and level of the weak constraint as follows:

Definition 9 (Objective Function, $H^{\Psi}(\mathcal{S})$, [7]). *Given a ground program Ψ with weak constraints $\Omega(\Psi)$ and an answer set $\mathcal{S}$, the ω* objective function *$H^{\Psi}(\mathcal{S})$ is defined by using an auxiliary function f_{Ψ} that maps levelled weights to weights without levels:*

$$\begin{aligned} f_{\Psi}(1) &= 1 \\ f_{\Psi}(n) &= f_{\Psi}(n-1) \cdot |\Omega(\Psi)| \cdot w^{\Psi}_{max} + 1, n > 1 \\ H^{\Psi}(\mathcal{S}) &= \sum_{i=1}^{l^{\Psi}_{max}} (f_{\Psi}(i) \cdot \sum_{\rho \in N^{\Psi}_i(\mathcal{S})} weight(\rho)) \end{aligned}$$

where w^{Ψ}_{max} and l^{Ψ}_{max} denote the maximum weight and maximum level over the weak constraints in Ψ, respectively; $N^{\Psi}_i(\mathcal{S})$ denotes the weak constraints at level i violated by $\mathcal{S}$, and $weight(\rho)$ the weight of weak constraint ρ.

Finally, the *best models* of such a logic program are those that minimise the number of violated weak constraints.

Definition 10 (Weak-Constraint Model, [7]). *For an EDLP Ψ with weak constraints, a set $\mathcal{S}$ is a* weak-constraint model *of Ψ if and only if*

1. *$\mathcal{S}$ is an answer set of $\Psi \setminus \Omega(\Psi)$*
2. *$H^{\Psi}(\mathcal{S})$ is minimal over all the answer sets of Ψ.*

where $\Omega(\Psi)$ is the finite set of weak constraints in Ψ.

When the underlying semantics is ASP in Definition 10, a weak-constraint model (or also ω-model) is also known as *Optimal Answer Set*. Moreover, the language of EDLP's with weak constraints shall be called $\mathsf{DATALOG}^{\vee,\omega}$, which is very similar to the notation from the literature.

2.4 Object-level Updates

Now let us briefly introduce the semantics to propose a preference characterisation and implementation, by the following definitions, taken from [1]:

Formally, an *α-relaxed rule* is a rule ρ that is weakened by a default-negated atom α in its body: $\mathsf{Head}(\rho) \leftarrow \mathsf{Body}(\rho) \cup \{\neg\alpha\}$. In addition, an *$\alpha$-relaxed program* is a set of α-relaxed rules. A *generalised program* of $\mathcal{A}^*$ is a set of rules of form $\{\ell \leftarrow \top \mid \ell \in \mathcal{A}^*\}$, where $\mathcal{A}^*$ is a given set of literals.

Definition 11 ($\bullet$-update Program, [1]). *Given an update pair of extended logic programs, denoted as $\Psi_1 \bullet \Psi_2$, over a set of atoms $\mathcal{A}$; and a set of unique abducibles $\mathcal{A}^*$, such that $\mathcal{A} \cap \mathcal{A}^* = \emptyset$; and the α-relaxed program Ψ' from Ψ_1, such that $\alpha \in \mathcal{A}^*$; and the abductive program $\Psi_{\mathcal{A}^*} = \langle \Psi' \cup \Psi_2, \mathcal{A}^* \rangle$. Its* $\bullet$-update program *is $\Psi' \cup \Psi_2 \cup \Psi_G$, where Ψ_G is a generalised program of $\mathcal{M} \cap \mathcal{A}^*$ for some* minimal generalised answer set $\mathcal{M}$ *of $\Psi_{\mathcal{A}^*}$ and "$\bullet$" is the corresponding update operator.*

Last, the associated *models $\mathcal{S}$ of the new knowledge base* correspond to the answer sets of a $\bullet$-update program as follows.

Definition 12 ($\bullet$-update Answer Set, [1]). *Let $\mathbf{\Psi}_\bullet = (\Psi_1 \bullet \Psi_2)$ be an update pair of extended logic programs over a set of atoms $\mathcal{A}$. Then, $\mathcal{S} \subseteq \mathcal{A}$ is a* $\bullet$-answer set of $\mathbf{\Psi}_\bullet$ *if and only if $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ for some* minimal generalised answer set $\mathcal{S}'$ *of its $\bullet$-update program.*

Having introduced all necessary background and notation to better understand the results of this proposal, let us present the main results in the following sections.

3 Preference and Debugging Characterisation

There are two well-known major efficient *solvers* to compute ASP with a vast background of implementation and research. They are DLV [7] and SMODELS [8], and the system proposed in this paper employs the former at a higher abstraction level in order to update ELP programs. This section is an introduction to a transformation that may be interpreted in DLV and possibly in any other ASP solver with some slight syntactic adaptations[2].

To begin with, an approach to implement an update semantics in MGAS was first suggested in [9] by means of preferred disjunctive logic programs in Brewka's ODLP semantics. However, the update semantics and thus the final transformation itself, are limited to single updates and no further details on its implementation are ever given. Not to mention that the latest version of the proposed solver endures quite a few bugs.

Indeed, a justification from [**?**] to use ODLP is that there is a solver available named PSMODELS[3] that is an extension to SMODELS [8] to compute *preferred answer sets*. Unfortunately, up to the printout of this paper, there is no reliable version of PSMODELS and the latest one (v.2.26a) endures some few *bugs* under certain circumstances[4]. In addition to the *running solver*, it is believed that DLV significantly outperforms SMODELS [7], not to mention that ODLP is such a colossal semantics that can do much more complex tasks than just computing MGAS's, which might compromise the performance of the desired system.

[2] Refer, for instance, to [7] for an *equivalence of weak constraints* in SMODELS.
[3] `http://www.tcs.hut.fi/Software/smodels/priority/`
[4] Try to compute the preferred models of a simple program like $\{a.\}$.

This section is an introduction to the use of DLV's Weak Constraints for their characteristics of *preferences* [7] that enjoys the above benefits of DLV with no more extra throughput added to the system.

3.1 Preferred Models in Weak Constraints

This approach consists in transforming updates of ELP's into a ω-program. The answer sets of such a program shall prove to coincide with the GAS's of the abductive program from the update. So, let us start by introducing some more notation.

A *Simple Weak Constraint* (ω') is an expression of the form

$$[w : p] \leftarrow \ell$$

where ℓ is a literal; and w and p are optional weight and level parameters as in Definition 7.

Before introducing a result derived from ω', let us introduce some new notation: read $\mathfrak{L}_{\Omega(\Psi)}$ as the *signature of the weak constraints* occurring in Ψ, which is a finite set of literals; while $\mathfrak{L}_{N_1^{\Psi}(\mathcal{M})}$ stands for the *signature* of the weak constraints in Ψ at level 1 violated by $\mathcal{M}$—also a finite set of literals. Something worth recalling is that $N_i^{\Psi}(\mathcal{M})$ denotes the weak constraints at level i violated by $\mathcal{M}$ in Ψ.

Proposition 1. *Suppose an EDLP, Ψ, with weak constraints $\Omega(\Psi)$ of the form of a* simple weak constraint*, with $w = p = 1$; suppose an answer set $\mathcal{M}$ of $\Psi \setminus \Omega(\Psi)$; and a set of literals $\Delta \subseteq \mathcal{M} \cap \mathfrak{L}_{\Omega(\Psi)}$. Then, $\mathfrak{L}_{N_1^{\Psi}(\mathcal{M})} = \Delta$.*

By a slight abuse of notation, let us say that the *model(s) of an EDLP with weak constraints* are also called answer sets.

Before introducing the translation function, let us extend the well-known standard definition of *signature* by $\mathfrak{L}_{\langle \Psi, \mathcal{A}^* \rangle}$, which means the finite set of literals occurring both in Ψ and in $\mathcal{A}^*$.

Definition 13 (Abductive-$\mathcal{W}$ Translation). *Let $\langle \Psi, \mathcal{A}^* \rangle$ be an abductive logic program and α' a unique literal, such that $\alpha' \notin \mathfrak{L}_{\langle \Psi, \mathcal{A}^* \rangle}$. A* translation into a weak-constraint program *$\mathcal{W}(\Psi, \mathcal{A}^*)$ corresponds to*

$$\Psi \cup \{\alpha' \vee \alpha \leftarrow \top, [1 : 1] \leftarrow \alpha \mid \alpha \in \mathcal{A}^*\}. \tag{3}$$

Both this translation and Proposition 1 yield the following useful results for the implementation of GAS-semantics in weak constraints.

Lemma 1. *$\mathcal{M} \cap \mathfrak{L}_{\langle \Psi, \mathcal{A}^* \rangle}$ is a* generalised answer set of an abductive program *$\langle \Psi, \mathcal{A}^* \rangle$ if and only if $\mathcal{M}$ is an answer set of $\mathcal{W}(\Psi, \mathcal{A}^*)$.*

Proof (sketch). Only-if part. Suppose $\mathcal{M}$ is an answer set of $\mathcal{W}(\Psi, \mathcal{A}^*)$. $\mathcal{M}$ is an answer set of $\Psi \cup \{\alpha' \vee \alpha \leftarrow \top, [1 : 1] \leftarrow \alpha\}$ with $\alpha \in \mathcal{M}$ and $\mathcal{M} \cap \mathfrak{L}_{\langle \Psi, \mathcal{A}^* \rangle}$ corresponds to the answer set of $\Psi \cup \{\alpha \leftarrow \top\}$ and therefore to the GAS of $\langle \Psi, \mathcal{A}^* \rangle$.

On the other hand, with $\alpha \notin \mathcal{M}$, $\mathcal{M} \cap \mathfrak{L}_{\langle \Psi, \mathcal{A}^* \rangle}$ is just an answer sets of $\Psi \cup \{\}$ and thus the GAS of $\langle \Psi, \mathcal{A}^* \rangle$. In both cases, $\mathcal{M} \cap \mathfrak{L}_{\langle \Psi, \mathcal{A}^* \rangle}$ is a GAS of $\langle \Psi, \mathcal{A}^* \rangle$, as required.

The if-part is similarly straightforward.

The *equivalence between a GAS of an abductive program and an answer set of a ω-program* ought to be easier to read after a simple example:

Example 1. Suppose an update of Ψ_1 with Ψ_2 where $\Psi_1 = \{b \leftarrow \top\}; \Psi_2 = \{a \leftarrow \top\}$. Its corresponding abductive program is $\langle \Psi' \cup \Psi_2, \mathcal{A}^* \rangle$ where $\Psi' = \{b \leftarrow \neg\alpha\}$ and $\mathcal{A}^* = \{\alpha\}$. As a result, $\mathcal{W}(\Psi' \cup \Psi_2, \mathcal{A}^*) = \Psi' \cup \Psi_2 \cup \{\alpha' \vee \alpha \leftarrow \top, [1:1] \leftarrow \alpha\}$ whose answer set $\mathcal{M} = \{\alpha', a, b\}$. On the other hand, the GAS of the abductive program is just $\{a, b\} = \mathcal{M} \cap \mathfrak{L}_{\langle \Psi' \cup \Psi_2, \mathcal{A}^* \rangle}$.

Up to now, one can compute GAS's by means of ω-programs. However, this proposal of *updates at the object level* requires that the generalised answer sets are minimal with respect to set inclusion. Thus, another result of this section is the following formalism that shows the *equivalence between MGAS's and inclusion-preferred answer sets.*

Definition 14 (Inclusion Preference). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be answer sets of a ω-program, $\mathcal{W}(\Psi, \mathcal{A}^*)$. Then, $\mathcal{M}_1$ is* inclusion-preferred to $\mathcal{M}_2$, denoted as $\mathcal{M}_1 \leq_{\mathcal{I}} \mathcal{M}_2$, if and only if $\mathfrak{L}_{N_1^{\Psi}(\mathcal{M}_1)} \subseteq \mathfrak{L}_{N_1^{\Psi}(\mathcal{M}_2)}$.

Corollary 1. *$\mathcal{M}$ is a* minimal generalised answer set *of $\langle \Psi, \mathcal{A}^* \rangle$ if and only if $\mathcal{M}$ is an answer set of the corresponding ω-program $\mathcal{W}(\Psi, \mathcal{A}^*)$, and is minimal with respect to its* inclusion preference.

With this equivalence, it is easy to deliver further results, linked to weak-constraints models, as follows.

Definition 15 (Cardinality Preference). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be answer sets of a ω-program, $\mathcal{W}(\Psi, \mathcal{A}^*)$. Then, $\mathcal{M}_1$ is* cardinality-preferred to $\mathcal{M}_2$, denoted as $\mathcal{M}_1 \leq_{\mathcal{C}} \mathcal{M}_2$, if and only if $|\mathfrak{L}_{N_1^{\Psi}(\mathcal{M}_1)}| \leq |\mathfrak{L}_{N_1^{\Psi}(\mathcal{M}_2)}|$.

Corollary 2. *$\mathcal{M}$ is a* minimal cardinality-preferred answer set *of $\langle \Psi, \mathcal{A}^* \rangle$ if and only if $\mathcal{M}$ is* weak-constraint model *of the corresponding ω-program $\mathcal{W}(\Psi, \mathcal{A}^*)$.*

This section is one of the main contribution of this paper and extends previous claims to implement the declarative semantics. First, it introduces a translation of an abductive logic program. Next, the translation has several ways to be interpreted and we propose two kinds of preferred models. Finally, we introduce the theoretical basis for its implementation.

The following section presents a general view of the implementation, which confirms our claims and provides a testbed with an online prototype[5] for further research, toy examples and a component of more complex applications.

[5] `http://fi.uaemex.mx/juan.acosta/kr-lab/od.html`.

3.2 Debugging ASP Programs

Another main contribution of this work is a general proposal to debug EDLP's through updates. This idea is similar to the one proposed by [2], who suggested a set of transformations to find the source of inconsistency. Such a proposal, however, has not been further developed.

By following the semantics introduced in Section 2.4, the rules in the $\bullet$-update program give enough information so as to find the source of conflict, by means of its α-rules, when one of them is satisfied. So, let us begin with some definitions.

A rule $\rho \in \Psi$ is said to *contradict* a consistent EDLP, Ψ, if $\{\rho\} \cup \Psi$ has no answer sets.

Proposition 2. *Suppose two consistent EDLP's Ψ_1, Ψ_2 and the update $\Psi_1 \bullet \Psi_2$ with its corresponding α-relaxed program, Ψ_1', its abductive program $\Psi_{\mathcal{A}^*}$ with a minimal generalised answer set $\mathcal{M}$; where $\Psi_1 \cup \Psi_2$ has no model. The rule $\rho \in \Psi_1$ contradicts Ψ_2 if and only if its corresponding α-relaxed rule $\rho' \in \Psi_1'$, where $\alpha \in \mathcal{M}$ and $\neg\alpha \in \mathit{Body}(\rho')$.*

Proof (sketch). The proof comes from the fact that Ψ_2 is consistent and an α-relaxed rule $\rho' \in \Psi_1'$ is inhibited when its corresponding abducible α is true.

As a result, one may find contradictory rules as in the following example.

Example 2. Suppose the ASP programs $\Psi_1 = \{(a \leftarrow \top), (b \leftarrow \neg c), (d \leftarrow \top\}$ and $\Psi_2 = \{{\sim}b \leftarrow \neg x\}$. Rule $(b \leftarrow \neg c)$ contradicts Ψ_2.

As opposed to Example 2, where Ψ_2 might be considered an update to Ψ_1, one may also debug a single (static) logic program, as in the following example.

Example 3. Suppose the inconsistent program

$$\{(a \leftarrow \top), (b \leftarrow \neg c), (d \leftarrow \top), ({\sim}b \leftarrow \top), ({\sim}a \leftarrow \neg x)\}$$

The following combinations of rules contradict: $(a \leftarrow \top), (b \leftarrow \neg c)$ or $(a \leftarrow \top), ({\sim}b \leftarrow \top)$ or $(b \leftarrow \neg c), ({\sim}a \leftarrow \neg x)$ or $({\sim}b \leftarrow \top), ({\sim}a \leftarrow \neg x)$.

Finally, the following result holds.

Proposition 3. *Suppose an inconsistent EDLP, Ψ_1, and the update $\Psi_1 \bullet \emptyset$ with its corresponding α-relaxed program, Ψ_1', its abductive program $\Psi_{\mathcal{A}^*}$ with a minimal generalised answer set $\mathcal{M}$. The rule $\rho \in \Psi_1$ contradicts Ψ_1 if and only if its corresponding α-relaxed rule $\rho' \in \Psi_1'$, where $\alpha \in \mathcal{M}$ and $\neg\alpha \in \mathit{Body}(\rho')$.*

This section is an introduction of the main results of this paper, which consist of a general characterisation of preferences in weak constraints and a method to debug logic programs in ASP. They provide a solid theoretical framework towards the implementation of a system in DLV to debug ASP knowledge bases.

4 Conclusions

In need of a general simpler accessible *integral framework* to debug knowledge bases, this paper is a *characterisation in terms of weak constraints* both for MGAS's and Optimal Answer Sets, of an updates semantics that provide a solid foundation for its implementation. The proposed prototype can be employed both to identify conflicts with upcoming information from a dynamic changing environment, and to locate the source of conflict from a given inherent inconsistent knowledge base that is static. In addition, this work introduces some formal specifications towards the implementation of a debugging solver with no need to use a total-order semantics.

References

[1] Acosta G., J. C. 2007. Maintaining knowledge bases at the object level. In *Special Session of the 6th International MICAI Conference*, A. Gelbukh and A. F. Kuri Morales, Eds. IEEE Computer Society, Aguascalientes, Mexico, 3–13. ISBN: 978-0-7695-3124-3.

[2] Acosta G., J. C., Arrazola, J., and Osorio, M. 2002. Making belief revision with LUPS. In *XI International Conference on Computing*, J. H. S. Azuela and G. A. Figueroa, Eds. Number ISBN: 970-18-8590-2. CIC-IPN, México, D.F.

[3] Balduccini, M. and Gelfond, M. 2003. Logic programs with consistency-restoring rules. In *Proceedings of the AAAI Spring 2003 Symposium*. AAAI Press, Palo Alto, California, 9–18.

[4] Brewka, G. 2002. Logic programming with ordered disjunction. In *Proceedings of the 18th National Conference on Artificial Intelligence, AAAI-2002*. Morgan Kaufmann, Edmonton, Alberta, Canada.

[5] Gelfond, M. and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium ICLP/SLP*, R. A. Kowalski and K. A. Bowen, Eds. MIT Press, Seattle, Washington, 1070–1080.

[6] Kakas, A. C. and Mancarella, P. 1990. Generalized Stable Models: A semantics for abduction. In *ECAI*. Stockholm, Sweden, 385–391.

[7] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562.

[8] Niemela, I. and Simons, P. 1997. Smodels—an implementation of the Stable Model and Well-Founded Semantics for normal logic programs. In *Proceedings of the 4th LPNMR ('97)*. LNCS, vol. 1265. Springer, Dagstuhl Castle, Germany, 420–429.

[9] Zacarías, F., Osorio, M., Acosta G., J. C., and Dix, J. 2005. Updates in Answer Set Programming Based on Structural Properties. In *7th International Symposium on Logical Formalizations of Commonsense Reasoning*, S. McIlraith, P. Peppas, and M. Thielscher, Eds. Fakultät Informatik, ISSN 1430-211X, Corfu, Greece, 213–219.