Revising Knowledge in Multi-agent Systems Using Revision Programming with Preferences

Inna Pivkina, Enrico Pontelli, and Tran Cao Son

Department of Computer Science,
New Mexico State University
{ipivkina, epontell, tson}@cs.nmsu.edu

Abstract. In this paper we extend the *Revision Programming* framework—a logic-based framework to express and maintain constraints on knowledge bases—with different forms of *preferences*. Preferences allow users to introduce a bias in the way agents update their knowledge to meet a given set of constraints. In particular, they provide a way to select one between alternative feasible revisions and they allow for the generation of revisions in presence of conflicting constraints, by relaxing the set of satisfied constraints (*soft constraints*). A methodology for computing preferred revisions using answer set programming is presented.

1 Introduction

Multi-Agents Systems (MAS) require coordination mechanisms to facilitate dynamic collaboration of the intelligent components, with the goal of meeting local and/or global objectives. In the case of MAS, the coordination structure should provide communication protocols to link agents having inter-related objectives and it should facilitate mediation and integration of exchanged knowledge [7]. Centralized coordination architectures (e.g., mediator-based architectures) as well as fully distributed architectures (e.g., distributed knowledge networks) face the problem of non-monotonically updating agent's theories to incorporate knowledge derived from different agents. The problem is compounded by the fact that incoming knowledge could be contradictory—either conflicting with the local knowledge or with other incoming items—incomplete, or unreliable. Recently a number of formalisms have been proposed [16, 4, 2, 20, 8] to support dynamic updates of (propositional) logic programming theories; they provide convenient frameworks for describing knowledge base updates as well as constraints to ensure user-defined principles of consistency. These types of formalisms have been proved effective in the context of MAS (e.g., [12]).

One of such formalisms for knowledge base updates is *Revision Programming*. Revision programming is a formalism to describe and enforce constraints on belief sets, databases, and more generally, on arbitrary knowledge bases. The revision programming formalism was introduced in [15, 16]. In this framework, the *initial database* represents the initial state of a belief set or a knowledge base. A *revision program* is a collection of *revision rules* used to describe constraints on the content of the database. Revision rules could be quite complex and are usually in the form of conditions. For instance, a typical revision rule may express a condition that, if certain elements are present in the database

and some other elements are absent, then another given element must be absent from (or present in) the database. Revision rules offer a natural way of encoding policies for the integration of agent-generated knowledge (e.g., in a mediator-based architecture) or for the management of inter-agent exchanges.

In addition to being a declarative specification of a constraint on a knowledge base, a revision rule also has a computational interpretation—indicating a way to satisfy the constraint. Justified revisions semantics assigns to any knowledge base a (possibly empty) family of *revisions*. Each revision represents an updated version of the original knowledge base, that satisfies all the constraints provided by the revision program. Revisions are obtained by performing additions and deletions of elements from the original knowledge base, according to the content of the revision rules. Each revision might be chosen as an update of the original knowledge base w.r.t. the revision program.

The mechanisms used by revision programming to handle updates of a knowledge base or belief set may lead to indeterminate situations. The constraints imposed on the knowledge base are interpreted as *hard constraints*, that have to be met at all costs; nevertheless this is rather unnatural in domains where overlapping and conflicting consistency constraints may be present (e.g., legal reasoning [18], suppliers and broker agents in a supply chain [13])—leading to the generation of *no* acceptable revisions. Similarly, situations with under-specified constraints or incomplete knowledge may lead to revision programs that provide *multiple* alternative revisions for the same initial knowledge base. While such situations might be acceptable, there are many cases where a single revision is desired—e.g., agents desire to maintain a unique view of a knowledge base.

Preferences provide a natural way to address these issues; preferences allow the revision programmer to introduce a bias, and focus the generation of revisions towards more desirable directions. Preferences between revisions rules and/or preferences between the components of the revisions can be employed to select the way revisions are computed, ruling out undesirable alternatives and defeating conflicting constraints. The use of preference structures has been gaining relevance in the MAS community as key mechanism in negotiation models for MAS coordination architectures [9, 11].

In this work we propose extensions of revision programming that provide general mechanisms to express different classes of preferences—justified by the needs of knowledge integration in MAS. The basic underlying mechanism common to the extensions presented in this work is the idea of allowing classes of revision rules to be treated as *soft revision rules*. A revision might be allowed even if it does not satisfy all the soft revision rules but only selected subsets of them; user preferences express criteria to select the desired subsets of soft revision rules.

Our first approach (Section 3) is based on the use of *revision programs with preferences*, where dynamic partial orders are established between the revision rules. It provides a natural mechanism to select preferred ways of computing revisions, and to prune revisions that are not deemed interesting. This approach is analogous to the ordered logic program (a.k.a. prioritized logic program) approach explored in the context of logic programming (e.g., [6, 5]). In a labeled revision program, the revision program and the initial knowledge base are enriched by a *control program*, which expresses preferences on rules. The control program may include revision literals as well as conditions on the initial knowledge base. Given an initial knowledge base, the control program

and the revision program are translated into a revision program where regular justified revisions semantics is used. This approach provides preference capabilities similar to those supported by the MINERVA agent architecture [12].

The second approach (Section 4) generalizes revision programs through the introduction of *weights* (or *costs*) associated to the components of a revision program (revision rules and/or database atoms). The weights are aimed at providing general criteria for the selection of subsets of the soft revision rules to be considered in the computation of the revisions of the initial database. Different policies in assigning weights are considered, allowing for the encoding of very powerful preference criteria (e.g., revisions that differ from the initial database in the least number of atoms). This level of preference management addresses many of the preference requirements described in the MAS literature (e.g., [11]).

For each of the proposed approaches to the management of preferences, we provide an effective implementation schema based on translation to answer set programming— specifically to the `smodels` [17] language. This leads to effective ways to compute *preferred* revisions for any initial database w.r.t. a revision program with preferences.

The main contribution of this work is the identification of forms of preferences that are specifically relevant to the revision programming paradigm and justified by the needs of knowledge maintenance and integration in MAS, and the investigation of the semantics and implementation issues deriving from their introduction.

2   Preliminaries: Revision Programming

In this section we present the formal definition of revision programs with justified revision semantics and some of their properties [16, 15, 14].

Elements of some finite universe U are called *atoms*. Subsets of U are called *databases*. Expressions of the form $\mathbf{in}(a)$ or $\mathbf{out}(a)$, where a is an atom, are called *revision literals*. For a revision literal $\mathbf{in}(a)$, its *dual* is the revision literal $\mathbf{out}(a)$. Similarly, the *dual* of $\mathbf{out}(a)$ is $\mathbf{in}(a)$. The dual of a revision literal α is denoted by α^D. A set of revision literals L is *coherent* if it does not contain a pair of dual literals. For any set of atoms $B \subseteq U$, we denote $B^c = \{\mathbf{in}(a) : a \in B\} \cup \{\mathbf{out}(a) : a \notin B\}$. A *revision rule* is an expression of one of the following two types:

$$\mathbf{in}(a) \leftarrow \mathbf{in}(a_1), \dots, \mathbf{in}(a_m), \mathbf{out}(b_1), \dots, \mathbf{out}(b_n) \qquad \text{or} \qquad (1)$$

$$\mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \dots, \mathbf{in}(a_m), \mathbf{out}(b_1), \dots, \mathbf{out}(b_n), \qquad (2)$$

where a, a_i and b_i are atoms. A *revision program* is a collection of revision rules. Revision rules have a declarative interpretation as constraints on databases. For instance, rule (1) imposes the following condition: a is *in* the database, or at least one a_i, $1 \leq i \leq m$, is *not* in the database, or at least one b_j, $1 \leq j \leq n$, is *in* the database.

Revision rules also have a computational (imperative) interpretation that expresses a way to enforce a constraint. Assume that all data items a_i, $1 \leq i \leq m$, belong to the current database, say I, and none of the data items b_j, $1 \leq j \leq n$, belongs to I. Then, to enforce the constraint (1), the item a must be added to the database (removed from it, in the case of the constraint (2)), rather than removing (adding) some item a_i (b_j).

Given a revision rule r, by $head(r)$ and $body(r)$ we denote the literal on the left hand side and the set of literals on the right hand side of the $\leftarrow$, respectively.

A set of atoms $B \subseteq U$ is a *model* of (or *satisfies*) a revision literal **in**(a) (resp., **out**(a)), if $a \in B$ (resp., $a \notin B$). A set of atoms B is a *model* of (or *satisfies*) a revision rule r if either B is not a model of at least one revision literal from the body of r, or B is a model of $head(r)$. A set of atoms B is a *model* of a revision program P if B is a model of every rule in P. Let P be a revision program. The *necessary change* of P, $NC(P)$, is the least model of P, when treated as a Horn program built of independent propositional atoms of the form **in**(a) and **out**(b).

The collection of all revision literals describing the elements that do not change their status in the transition from a database I to a database R is called the *inertia set* for I and R, and is defined as follows:

$$Inertia(I, R) = \{\mathbf{in}(a): a \in I \cap R\} \cup \{\mathbf{out}(a): a \notin I \cup R\}.$$

By the *reduct* of P with respect to a pair of databases (I, R), denoted by $P_{I,R}$, we mean the revision program obtained from P by eliminating from the body of each rule in P all literals in *Inertia*(I, R). The necessary change of the program $P_{I,R}$ provides a justification for some insertions and deletions. These are exactly the changes that are *a posteriori* justified by P in the context of the initial database I and a putative revised database R.

Given a database I and a coherent set of revision literals L, we define

$$I \oplus L = (I \setminus \{a \in U: \mathbf{out}(a) \in L\}) \cup \{a \in U: \mathbf{in}(a) \in L\}.$$

Definition 1 ([16]). *A database R is a P-justified revision of database I if the necessary change of $P_{I,R}$ is coherent and if $R = I \oplus NC(P_{I,R})$.*

Basic properties of justified revisions include the following [16]:

1. If a database R is a P-justified revision of I, then R is a model of P.
2. If a database B satisfies a revision program P then B is a unique P-justified revision of itself.
3. If R is a P-justified revision of I, then $R \div I$ is minimal in the family $\{B \div I : B$ is a model of $P\}$—where $R \div I$ denotes the symmetric difference of R and I. In other words, justified revisions of a database differ minimally from the database.

Another important property of revision programs is that certain transformations (*shifts*) preserve justified revisions [14]. For each set $W \subseteq U$, a W-*transformation* is defined as follows ([14]). If α is a literal of the form **in**(a) or **out**(a), then

$$T_W(\alpha) = \begin{cases} \alpha^D, & \text{when } a \in W \\ \alpha, & \text{when } a \notin W. \end{cases}$$

Given a set L of literals, $T_W(L) = \{T_W(\alpha): \alpha \in L\}$. For example, if $W = \{a, b\}$, then $T_W(\{\mathbf{in}(a), \mathbf{out}(b), \mathbf{in}(c)\}) = \{\mathbf{out}(a), \mathbf{in}(b), \mathbf{in}(c)\}$. Given a set A of atoms, $T_W(A) = \{a: \mathbf{in}(a) \in T_W(A^c)\}$. In particular, for any database I, $T_I(I) = \emptyset$. Given a revision program P, $T_W(P)$ is obtained from P by applying T_W to every literal in

P. The Shifting Theorem [14] states that for any databases I and J, database R is a P-justified revision of I if and only if $T_{I \div J}(R)$ is a $T_{I \div J}(P)$-justified revision of J. The Shifting Theorem provides a practical way [14] to compute justified revisions using answer set programming engines (e.g., `smodels` [17]). It can be done by executing the following steps.

1. Given a revision program P and an initial database I, we can apply the transformation T_I to obtain the revision program $T_I(P)$ and the empty initial database.
2. $T_I(P)$ can be converted into a logic program with constraints by replacing revision rules of the type (1) by

$$a \leftarrow a_1, \ldots, a_m, not\ b_1, \ldots, not\ b_n \tag{3}$$

and replacing revision rules of the type (2) by constraints

$$\leftarrow a, a_1, \ldots, a_m, not\ b_1, \ldots, not\ b_n. \tag{4}$$

We denote the logic program with constraints obtained from a revision program Q via the above conversion by $lp(Q)$.
3. Given $lp(T_I(P))$ we can compute its answer sets.
4. Finally, the transformation T_I can be applied to the answer sets to obtain the P-justified revisions of I.

3   Revision Programs with Preferences

In this section, we introduce *revision programs with preferences*, that can be used to deal with preferences between rules of a revision program. We begin with an example to motivate the introduction of preferences between revision rules. We then present the syntax and semantics and discuss some properties of revision programs with preferences.

3.1   Motivational Example

Assume that we have a number of agents $a_1, a_2, \ldots, a_n$. The environment is encoded through a set of parameters $p_1, p_2, \ldots, p_k$. The agents perceive parameters of the environment, and provide perceived data (observations) to a controller. The observations are represented using atoms of the form: $observ(Par, Value, Agent)$, where Par is the name of the observed parameter, $Value$ is the value for the parameter, and $Agent$ is the name of the agent providing the observation.

The controller combines the data received from agents to update its view of the world, which includes exactly one value for each parameter. The views of the world are described by atoms: $world(Par, Value, Agent)$, where $Value$ is the current value for the parameter Par, and $Agent$ is the name of the agent that provided the last accepted value for the parameter. The initial database contains a view of the world before the new observations arrive. A revision program, denoted by P, is used to update the view of the world, and is composed of rules of the type:

$$\mathbf{in}(observ(Par, Value, Agent)) \leftarrow$$

which describe all new observations; and rules of the following two types:

$$\mathbf{in}(world(Par, Value, Agent)) \leftarrow \mathbf{in}(observ(Par, Value, Agent)) \tag{a}$$

$$\mathbf{out}(world(Par, Value, Agent)) \leftarrow \mathbf{in}(world(Par, Value1, Agent1)),$$
$$(\text{where } Agent \neq Agent1 \ \ or \ \ Value \neq Value1). \tag{b}$$

Rules of type (a) allow to generate a new value for a parameter of a world view from a new observation. Rules of type (b) are used to enforce the fact that only one observation per parameter can be used to update the view.

It is easy to see that if the value of each parameter is perceived by *exactly one* agent and the initial world view of the controller is coherent, then each P-justified revision reflects the controller's world view that integrates its agent observations whenever they arrive. However, P does not allow any justified revisions when there are two agents which perceive different data for the same parameter at the same time. We illustrate this problem in the following scenario. Let us assume we have two agents a_1 and a_2, both provide observations for the parameter named *temperature* denoting the temperature in the room. Initially, the controller knows that $world(temperature, 76, s_2)$. At a later time, it receives two new observations

$$\mathbf{in}(observ(temperature, 74, a_1)) \leftarrow$$
$$\mathbf{in}(observ(temperature, 72, a_2)) \leftarrow$$

There is no P-justified revision for this set of observations as the necessary change with respect to it is incoherent, it includes $\mathbf{in}(world(temperature, 74, a_1))$ (because of (a) and the first observation) and $\mathbf{out}(world(temperature, 74, a_1))$ (because of (a), (b), and the second observation).

The above situation can be resolved by placing a preference between the values provided by the agents. For example, if we know that agent a_2 has a better temperature sensor than agent a_1, then we should tell the controller that observations of a_2 are preferred to those of a_1. This can be described by adding preferences of the form: *prefer*$(r2, r1)$, where $r1$ and $r2$ are names of rules of type (a) containing a_1 and a_2, respectively. With the above preference, the controller should be able to derive a justified revision which would contain $world(temperature, 72, a_2)$. If the agent a_2 has a broken temperature sensor and does not provide temperature observations, the value of *temperature* will be determined by a_1 and the world view will be updated correctly by P.

The above preference represents a fixed order of rule's application in creating revisions. Sometimes, preferences might be dynamic. As an example, we may prefer the controller to keep using temperature observations from the same agent if available. This can be described by preferences of the form:

prefer(r1,r2) $\leftarrow$ *world(temperature, Value, a_1)* $\in I$, $\mathbf{in}(observ(temperature, NewValue, a_1))$;
prefer(r2,r1) $\leftarrow$ *world(temperature, Value, a_2)* $\in I$, $\mathbf{in}(observ(temperature, NewValue, a_2))$;

where r1 and r2 are names of rules of type (a) containing a_1 and a_2 respectively, and I is an initial database (a view of the world before the new observations arrive).

3.2   Syntax and Semantics

A *labeled revision program* is a pair $(P, \mathcal{L})$ where P is a revision program and $\mathcal{L}$ is a function which assigns to each revision rule in P a unique name (label). The label of a rule $r \in P$ is denoted $\mathcal{L}(r)$. The rule with a label l is denoted $r(l)$. We will use $head(l)$, $body(l)$ to denote $head(r(l))$ and $body(r(l))$ respectively. The set of labels of all revision rules from P is denoted $\mathcal{L}(P)$. That is, $\mathcal{L}(P) = \{\mathcal{L}(r) : r \in P\}$. For simplicity, for each rule $\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n$ of P, we will write:

$$l : \alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n$$

to indicate that l is the value assigned to the rule by the function $\mathcal{L}$.

A *preference* on rules in $(P, \mathcal{L})$ is an expression of the following form

$$prefer(l_1, l_2) \leftarrow initially(\alpha_1, \ldots, \alpha_k), \alpha_{k+1}, \ldots, \alpha_n, \tag{5}$$

where l_1, l_2 are labels of rules in P, $\alpha_1 \ldots, \alpha_n$ are revision literals, $k \geq 0, n \geq k$.

Informally, the preference (5) mean that if revision literals $\alpha_1 \ldots, \alpha_k$ are satisfied by the initial database and literals $\alpha_{k+1}, \ldots, \alpha_n$ are satisfied by a revision, then we prefer to use rule $r(l_1)$ over rule $r(l_2)$. More precisely, if the body of rule $r(l_1)$ is satisfied then rule $r(l_2)$ is defeated and ignored. If $body(l_1)$ is not satisfied then rule $r(l_2)$ is used.

A *revision program with preferences* is a triple $(P, \mathcal{L}, S)$, where $(P, \mathcal{L})$ is a labeled revision program and S is a set of preferences on rules in $(P, \mathcal{L})$. We refer to S as the control program since it plays an important role on what rules can be used in constructing the revisions.

A revision program with preferences $(P, \mathcal{L}, S)$ can be translated into an ordinary revision program as follows. Let $U^{\mathcal{L}(P)}$ be the universe obtained from U by adding new atoms of the form $ok(l)$, $defeated(l)$, $prefer(l_1, l_2)$ for all $l, l_1, l_2 \in \mathcal{L}(P)$. Given an initial database I, we define a new revision program $P^{S,I}$ over $U^{\mathcal{L}(P)}$ as the revision program consisting of the following revision rules:

- for each $l \in \mathcal{L}(P)$, the revision program $P^{S,I}$ contains the two rules

$$head(l) \leftarrow body(l), \textbf{in}(ok(l)) \tag{6}$$

$$\textbf{in}(ok(l)) \leftarrow \textbf{out}(defeated(l)) \tag{7}$$

- for each preference $prefer(l_1, l_2) \leftarrow initially(\alpha_1, \ldots, \alpha_k), \alpha_{k+1}, \ldots, \alpha_n$ in S such that $\alpha_1 \ldots, \alpha_k$ are satisfied by I, $P^{S,I}$ contains the rules

$$\textbf{in}(prefer(l_1, l_2)) \leftarrow \alpha_{k+1}, \ldots, \alpha_n \tag{8}$$

$$\textbf{in}(defeated(l_2)) \leftarrow body(l_1), \textbf{in}(prefer(l_1, l_2)) \tag{9}$$

Following the compilation approach in dealing with preferences, we define the notion of $(P, \mathcal{L}, S)$-justified revisions of an initial database I as follows.

Definition 2. *A database R is a $(P, \mathcal{L}, S)$-justified revision of I if there exists $R' \subseteq U^{\mathcal{L}(P)}$ such that R' is a $P^{S,I}$-justified revision of I, and $R = R' \cap U$.*

The next example illustrates the definition of justified revisions with respect to revision programs with preferences.

Example 1. Let P be the program containing the rules

$$r_1 : \mathbf{in}(world(temperature, 76, a_1)) \leftarrow \mathbf{in}(observ(temperature, 76, a_1)).$$
$$r_2 : \mathbf{in}(world(temperature, 77, a_2)) \leftarrow \mathbf{in}(observ(temperature, 77, a_2)).$$

and the set S of preferences consists of a single preference $prefer(r_1, r_2)$. Let $I_1 = \{observ(temperature, 76, a_1), observ(temperature, 77, a_2)\}$ be the initial database. The revision program P^{S,I_1} is the following:

$$\mathbf{in}(world(temperature, 76, a_1)) \leftarrow \mathbf{in}(observ(temperature, 76, a_1)), \mathbf{in}(ok(r_1))$$
$$\mathbf{in}(world(temperature, 77, a_2)) \leftarrow \mathbf{in}(observ(temperature, 77, a_2)), \mathbf{in}(ok(r_2))$$
$$\mathbf{in}(ok(r_1)) \leftarrow \mathbf{out}(defeated(r_1))$$
$$\mathbf{in}(ok(r_2)) \leftarrow \mathbf{out}(defeated(r_2))$$
$$\mathbf{in}(prefer(r_1, r_2)) \leftarrow$$
$$\mathbf{in}(defeated(r_2)) \leftarrow \mathbf{in}(observ(temperature, 76, a_1)),$$
$$\mathbf{in}(prefer(r_1, r_2))$$

Since I_1 has only one P^{S,I_1}-justified revision,

$$R_1 = \left\{ \begin{array}{c} observ(temperature, 76, a_1), observ(temperature, 77, a_2), \\ world(temperature, 76, a_1), prefer(r_1, r_2), ok(r_1), defeated(r_2) \end{array} \right\},$$

then I_1 has only one $(P, \mathcal{L}, S)$-justified revision, $\{world(temperature, 76, a_1)\}$[1].
Now, consider the case where the initial database is

$$I_2 = \{observ(temperature, 77, a_2)\}.$$

The revision program $P^{S,I_2} = P^{S,I_1}$. Since I_2 has only one P^{S,I_2}-justified revision,

$$R_2 = \left\{ \begin{array}{c} world(temperature, 77, a_2), observ(temperature, 77, a_2), \\ prefer(r_1, r_2), ok(r_1), ok(r_2) \end{array} \right\},$$

we can conclude that I_2 has only one $(P, \mathcal{L}, S)$-justified revision,

$$\{world(temperature, 77, a_2)\}[1].$$

Notice the difference in the two cases: in the first case, rule r_2 is defeated and cannot be used in generating the justified revision. In the second case both rules can be used.

---

[1] We omit the observations from the revised database.

3.3   Properties

Justified revision semantics for revision programs with preferences extends justified revision semantics for ordinary revision programs. More precisely:

Theorem 1. *A database R is a $(P, \mathcal{L}, \emptyset)$-justified revision of I if and only if R is a P-justified revision of I.*

Proof. ($\Rightarrow$) Let R be a $(P, \mathcal{L}, \emptyset)$-justified revision of I. By definition, there exists $R' \subseteq U^{\mathcal{L}(P)}$ such that R' is a $P^{\emptyset, I}$-justified revision of I, and $R = R' \cap U$. By definition of a justified revision, $NC((P^{\emptyset, I})_{I, R'})$ is coherent, and $R' = I \oplus NC((P^{\emptyset, I})_{I, R'})$. Revision program $P^{\emptyset, I}$ consists of the rules of the form (6) and (7) only. Therefore, R' does not contain atoms of the form *defeated*(l) ($l \in \mathcal{L}(P)$). Thus, $(P^{\emptyset, I})_{I, R'}$ consists of rules

$$head(l') \leftarrow body(l'), \mathbf{in}(ok(l')) \qquad \qquad \text{(for all } l' \in P_{I,R})$$
$$\mathbf{in}(ok(l)) \leftarrow \qquad \qquad \text{(for all } l \in \mathcal{L}(P))$$

Hence, $NC((P^{\emptyset, I})_{I, R'}) = NC(P_{I,R}) \cup \{ok(l) : l \in \mathcal{L}(P)\}$. Since $NC((P^{\emptyset, I})_{I, R'})$ is coherent, $NC(P_{I,R})$ is coherent, too. If we take intersection with U of left- and right-hand sides of equation $R' = I \oplus NC((P^{\emptyset, I})_{I, R'})$, we get $R = I \oplus NC(P_{I,R})$. By definition, R is a P-justified revision of I.

($\Leftarrow$) Let R be a P-justified revision of I. Consider $R' = R \cup \{ok(l) : l \in \mathcal{L}(P)\}$. Let us show that R' is a $P^{\emptyset, I}$-justified revision of I. Indeed, *Inertia*(I, R') contains all revision literals of the form $\mathbf{out}(defeated(l))$. Therefore, $(P^{\emptyset, I})_{I, R'}$ consists of rules

$$head(l') \leftarrow body(l'), \mathbf{in}(ok(l')) \qquad \qquad \text{(for all } l' \in P_{I,R})$$
$$\mathbf{in}(ok(l)) \leftarrow \qquad \qquad \text{(for all } l \in \mathcal{L}(P))$$

Thus, $NC((P^{\emptyset, I})_{I, R'}) = NC(P_{I,R}) \cup \{\mathbf{in}(ok(l)) : l \in \mathcal{L}(P)\}$. Consequently, $I \oplus NC((P^{\emptyset, I})_{I, R'}) = (I \oplus NC(P_{I,R})) \cup \{ok(l) : l \in \mathcal{L}(P)\} = R \cup \{ok(l) : l \in \mathcal{L}(P)\} = R'$. By definition, R' is a $P^{\emptyset, I}$-justified revision of I. Hence, R is a $(P, \mathcal{L}, \emptyset)$-justified revision of I. $\qquad \square$

We will now investigate other properties of revision programs with preferences. Because of the presence of preferences, it is expected that not every $(P, \mathcal{L}, S)$-justified revision of I is a model of P. This can be seen in the next example.

Example 2. Let P be the program

$$r_1 : \mathbf{in}(a) \leftarrow \mathbf{out}(b) \qquad \qquad r_2 : \mathbf{in}(b) \leftarrow \mathbf{out}(a)$$

and the set S consists of two preferences: *prefer*(r_1, r_2) and *prefer*(r_2, r_1). Then, $\emptyset$ is $(P, \mathcal{L}, S)$-justified revision of $\emptyset$ (both rules are defeated) but not a model of P.

The above example also shows that circular preferences among rules whose bodies can be satisfied simultaneously, may lead to a situation when all such rules will defeat each other, and therefore, none of the rules involved will be used in computing justified revisions. This situation corresponds to a conflict among preferences. For instance, in the

above example a conflict is between a preference to use r_1 instead of r_2 and a preference to use r_2 instead of r_1. In order to satisfy the preferences both rules need to be removed.

The next theorem shows that for each $(P, \mathcal{L}, S)$-justified revision R of I, the subset of rules in P that are satisfied by R, is uniquely determined. To formulate the theorem, we need some more notation. Let J be a subset of $U^{\mathcal{L}(P)}$. By $P|_J$ we denote the program consisting of the rules r in P such that

- $ok(\mathcal{L}(r)) \in J$, or
- $ok(\mathcal{L}(r)) \notin J$ and $body(r) \setminus J^c \neq \emptyset$.

Theorem 2. *For every $P^{S,I}$-justified revision R of I, the corresponding $(P, \mathcal{L}, S)$-justified revision $R \cap U$ of I is a model of program $P|_R$.*

Proof. Consider a rule r in $P|_R$. Let us prove that $R \cap U$ is a model of r. If $body(r)$ is not satisfied by $R \cap U$, then r is trivially satisfied by $R \cap U$. Assume that $body(r)$ is satisfied by $R \cap U$. Since all revision literals in $body(r)$ belong to U^c, $body(r)$ is satisfied by R, and $body(r) \setminus R^c = \emptyset$. By definition of $P^{S,I}$, rule r'

$$r' = head(r) \leftarrow body(r), \mathbf{in}(ok(\mathcal{L}(r)))$$

belongs to $P^{S,I}$. By definition of $P|_R$, $ok(\mathcal{L}(r)) \in R$. Hence, $body(r')$ is satisfied by R. By definition of a (P, L, S)-justified revision, R is a model of $P^{S,I}$. Therefore, $head(r') = head(r)$ is satisfied by R. Since $head(r) \in U^c$, it is satisfied by $R \cap U$. Thus, $R \cap U$ is a model r. Consequently, $R \cap U$ is a model of $P|_R$. □

In the rest of this subsection, we discuss some properties that guarantee that each (P, L, S)-justified revision of I is a model of the program P. We concentrate on conditions on the set of preferences S. Obviously, Example 2 suggests that S should not contain a cycle between rules. The next example shows that if preferences are placed on a pair of rules such that the body of one of them is satisfied when the other rule is fired, then this may result in revisions that are not models of the program.

Example 3. Let P be the program

$$r_1 : \mathbf{in}(a) \leftarrow \mathbf{in}(b) \qquad r_2 : \mathbf{in}(d) \leftarrow \mathbf{out}(a)$$

and the set of preferences S consists of $prefer(r_2, r_1)$. Then, $\{b, d\}$ is $(P, \mathcal{L}, S)$-justified revision of $\{b\}$ but is not a model of P.

We now define precisely the conditions that guarantee that justified revisions of revision programs with preferences are models of the revision programs as well. First, we define when two rules are disjoint, i.e., when two rules cannot be used at the same time in creating revisions.

Definition 3. *Let $(P, \mathcal{L}, S)$ be a revision program with preferences. Two rules r, r' of P are* disjoint *if one of the following conditions is satisfied:*

1. *$(head(r))^D \in body(r')$ and $(head(r'))^D \in body(r)$; or*
2. *$body(r) \cup body(r')$ is incoherent.*

We say that a set of preferences is *selecting* if it contains only preferences between disjoint rules.

Definition 4. *Let $(P, \mathcal{L}, S)$ be a revision program with preferences. S is said to be a set of* selecting preferences *if for every preference*

$$prefer(r, r') \leftarrow l_1, \ldots, l_k$$

in S, rules r and r' are disjoint.

Finally, we say that a set of preferences is cycle-free if the transitive closure of the preference relation *prefer* does not contain a cycle.

Definition 5. *Let $(P, \mathcal{L}, S)$ be a revision program with preferences and $<_S = \{(r_1, r_2) \mid prefer(r_1, r_2)$ occurs as head of a preference in S and $(body(r_1) \cup body(r_2))$ is coherent$\}$. S is said to be* cycle-free *if for every rule r of P, (r, r) does not belong to the transitive closure $<_S^*$ of $<_S$.*

Lemma 1. *Let $(P, \mathcal{L}, S)$ be a revision program with preferences where S is a set of selecting preferences. Let R be a $(P, \mathcal{L}, S)$-justified revision of I. For every rule r in P such that $head(r) \notin R^c$ and $body(r) \subseteq R^c$ there exists a rule r' such that $(r', r) \in <_S$, $head(r') \notin R^c$, and $body(r') \subseteq R^c$.*

Proof. Let $R' \subseteq U^{\mathcal{L}(P)}$ be a $P^{S,I}$-justified revision of I such that $R = R' \cap U$ (it exists by definition of a $(P, \mathcal{L}, S)$-justified revision). Because $head(r) \notin R^c$ and $body(r) \subseteq R^c$, we have that $defeated(r) \in R'$. Hence, there exists a rule r' in P and a preference

$$prefer(r', r) \leftarrow l_1, \ldots, l_k$$

in S such that $\{l_1, \ldots, l_k\} \subseteq R^c$ and $body(r') \subseteq R^c$. Since $body(r) \subseteq R^c$ and $body(r') \subseteq R^c$, the set $(body(r') \cup body(r))$ is coherent. Therefore, $(r', r) \in <_S$. Rules r and r' are disjoint because S is a set of selecting preferences. Condition 2 in the definition of disjoint rules for r and r' is not satisfied because $(body(r') \cup body(r))$ is coherent. Hence, condition 1 must be satisfied. Namely, $(head(r))^D \in body(r')$ and $(head(r'))^D \in body(r)$. Because $body(r) \subseteq R^c$ and R^c does not contain a pair of dual literals, we conclude that $head(r') \notin R^c$. This proves the lemma.   □

The next theorem shows that the conditions on the set of preferences S guarantee that preferred justified revisions are models of the original revision program.

Theorem 3. *Let $(P, \mathcal{L}, S)$ be a revision program with preferences where S is a set of selecting preferences and is cycle-free. For every $(P, \mathcal{L}, S)$-justified revision R of I, R is a model of P.*

Proof. Let r be a rule in P. If $body(r)$ is not satisfied by R then rule r is trivially satisfied by R. Assume that $body(r)$ is satisfied by R. That is, $body(r) \subseteq R^c$. We need to prove that in this case $head(r) \in R^c$. Assume the contrary, $head(r) \notin R^c$. By Lemma 1, we know that there exists a rule r_1 such that $(r_1, r) \in <_S$, $body(r_1) \subseteq R^c$, and $head(r_1) \notin R^c$. Applying Lemma 1 one more time, we conclude that there exists a

rule r_2 such that $(r_2, r_1) \in <_S, body(r_2) \subseteq R^c$, and $head(r_2) \notin R^c$, etc. In other words, this implies that there exists an infinite sequence $r_0 = r, r_1, \ldots, r_k, r_{k+1}, \ldots$ such that $(r_{j+1}, r_j) \in <_S$. Since P is finite, we can conclude that there exists some $t > s$ such that $r_t = r_s$. This implies that $(r_t, r_t) \in <_S^*$, i.e., S is not cycle-free. This contradicts the assumption that S is cycle-free. In other words, our assumption that $head(r) \notin R^c$ is wrong. This proves the theorem. $\square$

The next theorem discusses the shifting property of revision programs with preferences. We extend the definition of W-*transformation* to a set of preferences on rules. Given a preference on rules p of the form (5), its W-transformation is the preference

$$T_W(p) = prefer(l_1, l_2) \leftarrow initially(T_W(\alpha_1), \ldots, T_W(\alpha_k)), T_W(\alpha_{k+1}), \ldots, T_W(\alpha_n).$$

Given a set of preferences S, its W-transformation is $T_W(S) = \{T_W(p) : p \in S\}$.

Theorem 4. *Let $(P, \mathcal{L}, S)$ be a revision program with preferences. For every two databases I_1 and I_2, a database R_1 is a $(P, \mathcal{L}, S)$-justified revision of I_1 if and only if $T_{I_1 \div I_2}(R_1)$ is a $(T_{I_1 \div I_2}(P), \mathcal{L}, T_{I_1 \div I_2}(S))$-justified revision of I_2.*

Proof. Let $W = I_1 \div I_2$.
($\Rightarrow$) Let R_1 be a $(P, \mathcal{L}, S)$-justified revision of I_1. By definition, there exists R_1' such that R_1' is a P^{S, I_1}-justified revision of I and $R_1 = R_1' \cap U$. It is straightforward to see that $T_W(P^{S, I_1}) = T_W(P)^{T_W(S), I_2}$. This together with the Shifting Theorem [14] implies that $T_W(R_1')$ is a $T_W(P)^{T_W(S), I_2}$-justified revision of $T_W(I_1) = I_2$. Notice that $T_W(R_1') \cap U = T_W(R_1)$. Therefore, $T_W(R_1)$ is a $(T_W(P), \mathcal{L}, T_W(S))$-justified revision of I_2.
($\Leftarrow$) The proof in the other direction is similar. $\square$

4   Soft Revision Rules with Weights

Preferences between rules (Section 3) can be useful in at least two ways. They can be used to recover from incoherency when agents provide inconsistent data, as in the example from Section 3.1. They can also be used to eliminate some revisions. The next example shows that in some situations, this type of preferences is rather weak.

Example 4. Consider again the example from Section 3.1, with two agents a_1 and a_2 whose observations are used to determine the value of the parameter *temperature*. Let us assume now that a_1 and a_2 are of the same quality, i.e., *temperature* can be updated by one of the observations yielded by a_1 and a_2. This means that there is no preference between the rule of type (a) (for a_1) and the rule of type (a) (for a_2) and vice versa. Yet, as we can see, allowing both rules to be used in computing the revisions will not allow the controller to update its world view when the observations are inconsistent.

The problem in the above example could be resolved by grouping the rules of the type (a) into a set and allowing only one rule from this set to be used in creating revisions if the presence of all the rules does not allow justified revisions.
Inspired by the research in constraint programming, we propose to address the situation when there are no justified revisions by dividing a revision program P in two

parts, HR and SR, i.e., $P = HR \cup SR$. Rules from HR and SR are called *hard rules* and *soft rules*, respectively. The intuition is that rules in HR must be satisfied by each revision, while revisions may satisfy only a subset of SR if it is impossible to satisfy all of them. The subset of soft rules that is satisfied, say S, should be optimal with respect to some comparison criteria. In this section, we investigate several criteria—each one is discussed in a separate subsection.

4.1   Maximal Number of Rules

Let $P = HR \cup SR$. Our goal is to find revisions that satisfy all rules from HR and the most number of rules from SR. Example 4 motivates the search for this type of revisions. In the next definition, we make this precise.

Definition 6. *R is a (HR, SR)-preferred justified revision of I if R is a $(HR \cup S)$ - justified revision of I for some $S \subseteq SR$, and for all $S' \subseteq SR$ such that S' has more rules than S, there are no $(HR \cup S')$-justified revisions of I.*

Preferred justified revision can be computed, under the maximal number of rules criteria, by extending the translation of revision programs to answer set programming, to handle the distinction between hard and soft rules. The objective is to determine $(HR \cup S)$-justified revisions of an initial database I, where S is a subset of SR of maximal size such that $(HR \cup S)$-justified revisions exist.

The idea is to make use of two language extensions proposed by the smodels system: choice rules and maximize statements. Intuitively, each soft rule can be either accepted or rejected in the program used to determine revisions. Let us assume that the rules in $T_I(SR)$ have been uniquely numbered. For each initial database I, we translate $P = HR \cup SR$ into an smodels program $lp(T_I(HR)) \cup lp'(T_I(SR))$ where $lp'(T_I(SR))$ is defined as follows. If the rule number i in $T_I(SR)$ is

$$\mathbf{in}(a) \leftarrow \mathbf{in}(p_1), \dots, \mathbf{in}(p_m), \mathbf{out}(s_1), \dots, \mathbf{out}(s_n)$$

then the following rules are added to $lp'(T_I(SR))$

$$
\begin{aligned}
\{rule_i\} &: - \, p_1, \dots, p_m, not \, s_1, \dots, not \, s_n. \\
a \quad &: - \, rule_i
\end{aligned}
$$

where $rule_i$ is a distinct new atom. Similarly, if

$$\mathbf{out}(a) \leftarrow \mathbf{in}(p_1), \dots, \mathbf{in}(p_m), \mathbf{out}(s_1), \dots, \mathbf{out}(s_n)$$

is the rule number i in $T_I(SR)$, then the following rules are added to $lp'(T_I(SR))$

$$
\begin{aligned}
\{rule_i\} &: - \, p_1, \dots, p_m, not \, s_1, \dots, not \, s_n. \\
&: - \, rule_i, a.
\end{aligned}
$$

where $rule_i$ is a distinct new atom. Finally, we need to enforce the fact that we desire to maximize the number of SR rules that are satisfied. This corresponds to maximizing the number of $rule_i$ that are true in the computed answer sets. This can be directly expressed by the following statement:

$$\mathtt{maximize}\{rule_1, \dots, rule_k\}. \tag{10}$$

where k is the number of rules in SR. The way how `smodels` system processes `maximize` statement is as follows. It first searches a single model and prints it. After that, `smodels` prints only "better" models. The last model that `smodels` prints will correspond to a (HR, SR)-preferred justified revision of I. Notice that this is the only occurrence of `maximize` in the translation which is a requirement for the correct handling of this construct in `smodels`.

4.2   Maximal Subset of Rules

A variation of definition from Section 4.1 can be obtained when instead of satisfying maximal number of soft rules it is desired to satisfy a maximal subset of soft rules. In other words, given $P = HR \cup SR$, the goal is to find revisions that satisfy all rules from HR and a maximal subset (with respect to set inclusion) of rules from SR. The precise definition follows.

Definition 7. *R is a (HR, SR)-preferred$^{\subseteq}$ justified revision of I if R is a $(HR \cup S)$ - justified revision of I for some $S \subseteq SR$, and for all S' if $S \subset S' \subseteq SR$, then there are no $(HR \cup S')$-justified revisions of I.*

The procedure described in Section 4.1 allows to compute only one of (HR, SR)-preferred$^{\subseteq}$ justified revisions which has maximal number of soft rules satisfied.

4.3   Weights

An alternative to the maximal subset of soft rules is to assign weights to the revisions and then select those with the maximal (or minimal) weight. In this section, we consider two different ways of assigning weights to revisions. First, we assign weight to rule. Next, we assign weight to atoms. In both cases, the goal is to find a subset S of SR such that the program $HR \cup S$ has revisions whose weight is maximal.

Weighted Rules. Each rule r in SR is assigned a weight (a number), $w(r)$. Intuitively, $w(r)$ represents the importance of r, i.e., the more the weight of a rule the more important it is to satisfy it.

Example 5. Let us reconsider the example from Section 3.1. Rules of the type (a) are treated as soft rules, while the rules of type (b) are treated as hard rules. We can make use of rule weights to select desired revisions. For example, if an observed parameter value falls outside expected value range for the parameter, it may suggest that an agent that provided the observation has a faulty sensor. Thus, we may prefer observations that are closer to the expected range. This can be expressed by associating to each rule r of the type (a) the weight

$$w(r) = min\{0, MaxEV - Value\} + min\{0, Value - MinEV\},$$

where $MaxEV$ and $MinEV$ are maximum and minimum expected values for Par.

Let us define the rule-weighted justified revision of a program with weights for rules.

Definition 8. *R is called a rule-weighted (HR, SR)-justified revision of I if the following two conditions are satisfied:*

1. *there exists a set of rules $S \subseteq SR$ s.t. R is a $(HR \cup S)$-justified revision of I, and*
2. *for any set of rules $S' \subseteq SR$, if R' is a $(HR \cup S')$-justified revision of I, then the sum of weights of rules in S' is less or equal than the sum of weights of rules in S.*

Let us generalize the implementation in the previous sections to consider weighted rules. The underlying principle is similar, with the difference that the selection of soft rules to include is driven by the goal of maximizing the total weight of the soft rules that are satisfied by the justified revision. The only change we need to introduce w.r.t. the implementation is in the `maximize` statement. Let us assume that $w(i)$ denotes the weight associated to the ith SR rule. Then, instead of the rule (10) the following maximize statement is generated:

$$\texttt{maximize}[rule_1 = w(1), rule_2 = w(2), \dots, rule_k = w(k)].$$

Weighted Atoms. Instead of assigning weights to rules, we can also assign weights to atoms in the universe U. Each atom a in the universe U is assigned a weight $w(a)$ which represented the degree we would like to keep it unchanged, i.e., the more the weight of an atom the less we want to change its status in a database. The next example presents a situation where this type of preferences is desirable.

Example 6. Let us return to the example from Section 3.1 with the same partition of rules in hard and soft rules as in Example 5. Let us assume that the choice of which observation to use to update the view of the world is based on the principle that stronger values for the parameters are preferable, as they denote a stronger signal. This can be encoded by associating weights of the form

$$w(world(Param, Value, Sensor)) = -Value$$

and minimizing the total weight of the revision.

Let us define preferred justified revision for programs with weight atoms.

Definition 9. *R is called an atom-weighted (HR, SR)-justified revision of I if the following two conditions are satisfied:*

1. *there exists a set of rules $S \subseteq SR$ s.t. R is a $(HR \cup S)$-justified revision of I, and*
2. *for any set of rules $S' \subseteq SR$, if Q is a $(HR \cup S')$-justified revision of I, then the sum of weights of atoms in $I \div Q$ is greater than or equal to the sum of weights of atoms in $I \div R$.*

Atom-weighted revisions can be computed using `smodels`. In this case the selection of the SR rules to be included is indirectly driven by the goal of minimizing the total weight of the atoms in $I \div R$, if I is the initial database and R is the justified revision. We make use of the following observation: given a revision program P and an initial database I, if $T_I(R)$ is a $T_I(P)$-justified revision of $\emptyset$, then $T_I(R) = I \div R$. Thanks to this observation, the computation of the weight of the difference between the original database and a P-justified revision can be computed by determining the *total* weight of the true atoms obtained from the answer set generated for the $T_I(P)$ program.

The program used to compute preferred revisions is encoded similarly to what is done for the case of maximal number of rules or weighted rules, i.e., each soft rule is encoded using `smodels`'s' choice rules. The only difference is that instead of making use of a `maximize` statement, we make use of a `minimize` statement of the form:

$$\texttt{minimize}[a_1 = w(a_1), a_2 = w(a_2), \ldots, a_n = w(a_n)] \tag{11}$$

where $a_1, \ldots, a_n$ are all the atoms in U.

4.4   Minimal Size Difference

In this subsection, we consider justified revisions that have minimal size difference with initial database. The next example shows that this is desirable in different situations.

Example 7. Assume that a department needs to form a committee to work on some problem. Each of the department faculty members has his or her own conditions on the committee members which need to be satisfied. The head of the department provided an initial proposal for members of the committee. The task is to form a committee which will satisfy all conditions imposed by the faculty members and will differ the least from the initial proposal — the size of the symmetric difference between the initial proposal and its revision is minimal.

In this problem we have a set of agents (faculty members) each of which provides its set of requirements (revision rules). The goal is to satisfy all agent's requirements in such a way that the least number of changes is made to the initial database (proposal).

Assume that faculty members are Ann, Bob, Chris, David, Emily and Frank. Conditions that they impose on the committee are the following:

$$
\begin{aligned}
Ann: \quad \mathbf{in}(Bob) \quad &\leftarrow \mathbf{out}(Chris) \\
\mathbf{in}(Chris) \quad &\leftarrow \mathbf{out}(Bob) \\
Bob: \mathbf{out}(David) &\leftarrow \mathbf{in}(Bob) \\
Chris: \quad \mathbf{out}(Ann) \quad &\leftarrow \mathbf{out}(David) \\
David: \mathbf{in}(David) \quad &\leftarrow \mathbf{in}(Chris), \mathbf{out}(Ann)
\end{aligned}
$$

The initial proposal is $I = \{Ann, David\}$. Then, there is one minimal size difference P-justified revisions of I, which is $R_1 = \{Ann, David, Bob\}$. The size of the difference $R_1 \div I$ is 1.

Ordinary P-justified revisions of I also include $R_2 = \{Bob\}$ with size of the difference $R_2 \div I$ equal to 3.

The next definition captures what is a minimal size different justified revision.

Definition 10. *R is called a minimal size difference P-justified revision of I if the following two conditions are satisfied:*

1. *R is a P-justified revision of I, and*
2. *for any P-justified revision R', the number of atoms in $R \div I$ is less than or equal to the number of atoms in $R' \div I$.*

Minimal size difference justified revision can be computed in almost the same way as for atom-weighted justified revisions. The intuition is that instead of minimizing the

total weight of $I \div R$ (where I is the initial database and R is a P-justified revision), we would like to minimize the size of $I \div R$. This can be accomplished by replacing the `minimize` statement (11) with the following `minimize` statement:

$$\texttt{minimize}\{a_1, a_2, \ldots, a_n\}$$

where $a_1, \ldots, a_n$ are all the atoms in U.

5   Related Work

Since revision programming is strongly related to the logic programming formalisms [15, 14, 19], our work is related to several works on reasoning with preferences in logic programming. In this section, we discuss the differences and similarities between our approach and some of the research in this area. In logic programming, preferences have been an important source for "correct reasoning". Intuitively, a logic program is developed to represent a problem, with the intention that its semantics (e.g., answer set or well-founded semantics) will yield correct answers to the specific problem instances. Adding preferences between rules is one way to eliminate counter-intuitive (or unwanted) results. Often, this also makes the program easier to understand and more elaboration tolerant. In the literature on logic programming with preferences, we can find at least two distinct ways to handle preferences. The first approach is to compile the preferences into the program (e.g., [10, 6]): given a program P with a set of preferences *pref*, a new program P_{pref} is defined whose answer set semantics is used as the preferred semantics of P with respect to *pref*. The second approach deals with preferences between rules by defining a new semantics for logic programs with preferences (e.g., [5, 21]). The advantage of the first approach is that it does not require the introduction of a new semantics — thus, answer set solvers can be used to compute the preferred semantics. The second approach, on the other hand, provides a more direct treatment of preferences.

    Section 3 of this paper follows the first approach. We define a notion of *revision program with preferences*, which is a labeled revision program with preferences between the rules. Given a revision program with preferences, we translate it into an ordinary revision program, and we define justified revisions w.r.t. the revision program with preferences as justified revisions w.r.t. the revision program obtained by translation. Our treatment of preferences is similar to that in [10, 6, 1]. In section 4, we introduce different types of preferences that can be dealt with more appropriately by following the second approach.

    We will now discuss the relationship between our approach and others in greater detail. We will compare revision programs with preferences with ordered choice logic programs [21] and preferred answer sets [5]. Both frameworks allow preferences between rules — similar to our *prefer* relation — to be added to programs (choice logic programs [21], and extended logic programs [5]). The main difference between our approach and the approaches in [5, 21] lies in that we adopt the compilation approach while preferences in [5, 21] are dealt with using the second approach.

    Ordered choice logic programs are introduced in [21] for modeling decision making with dynamic preferences. An ordered choice logic program (OCLP) P is a pair $(C, \preceq)$ where C is a set of choice logic programs whose rules are of the form $A \leftarrow B$ where

A and B are finite sets of atoms and $\preceq$ is a partial order on C. Intuitively, atoms in A represent alternatives and are assumed to be xor'ed together. Each member of C is called a component of P. Intuitively, $\preceq$ specifies an order in which the components of P are preferred. This ordering is used to select rules that can be applied to generate stable models of P. Given an interpretation I, a rule r is defeated with respect to I if there exist(s) some not less preferred rule(s) that can be applied in I whose head(s) contain(s) alternatives to the literals in the head of r. The stable model semantics of OCLP is defined in the same fashion of the original stable model semantics, i.e., given an interpretation M of P, a reduction of P with respect to M – which is a positive logic program – is defined; and, M is a stable model of P iff M is the stable model of the reduction of P with respect to M. It is worth noticing that in the first step of the reduction, defeated rules with respect to M are removed from P. The syntax difference between OCLP and revision program with preferences does not allow a detailed comparison between the two approaches. However, we note that OCLP follows the second approach to deal with preferences while our revision program with preferences uses the compilation approach. It is also interesting to notice that when the head of every rule in a OCLP program P has exactly one element then the preference order does not make any difference in computing stable models of P since there are no defeated rules. This could lead to a situation where P has a stable model M and P contains two rules, r and r', which belong to two components P_i and P_j, respectively, P_j is more specific than P_i, bodies of both r and r' are satisfied in M, and both r and r' are fired. Our formalization makes sure that this situation never happens (due to (6) and (7)). For example, consider the program $P = (C, \preceq)$ with

$$C = \{P_1, P_2\}, \ P_1 = \{p \leftarrow\}, \ P_2 = \{q \leftarrow\}, \ and \ \preceq = \{P_1 \prec P_2\}.$$

Then, $\{p, q\}$ is a stable model of this program. On the other hand, the corresponding revision program with preferences $(P', \mathcal{L}, S)$ with

$$P' = \{r_1 : \mathbf{in}(p) \leftarrow, \ r_2 : \mathbf{in}(q) \leftarrow\}, \ and \ S = \{prefer(r_1, r_2)\}$$

has only $\{p\}$ as its unique $(P', \mathcal{L}, S)$-justified revision of $\emptyset$.

In [5], preferred answer sets for prioritized logic programs with preferences between rules are defined. A new semantics is introduced that satisfies the two principles for priorities: one represents a meaning postulate for the term "preference" and the other is related to relevance. A detailed discussion on the differences and similarities between preferred answer sets for prioritized logic programs and other approaches to preferences handling in logic programming can be found in [5]. For a prioritized logic programs $(P, <)$, where P is an extended logic program and $<$ is a preference ordering between rules of P, the semantics in [5] requires that if A is a preferred answer set of $(P, <)$ then A is an answer set of P. Furthermore, A is generated by applying the rules in the order specified by $<$. Because this is not a requirement in compilation approach, it is not surprising to see that the approach we have taken to deal with preferences in labeled revision programs yield different results comparing to preferred answer sets. For example, consider the program $(P, <)$ with

$$P = \{r_1 : p \leftarrow \ not \ q, \ r_2 : q \leftarrow\}, \ and \ <= \{r_1 < r_2\}.$$

Then, $(P, <)$ does not have a preferred answer set because its only answer set $\{q\}$ cannot be generated by first applying the rule r_1 and then the rule r_2. On the other hand, the corresponding labeled program $(P', \mathcal{L}, S)$ with

$$P' = \{r_1 : \mathbf{in}(p) \leftarrow \mathbf{out}(q), \ r_2 : \mathbf{in}(q) \leftarrow\}, \ and \ S = \{prefer(r_1, r_2)\}$$

will have only $\{p\}$ as its unique $(P', \mathcal{L}, S)$-justified revision of $\emptyset$ because rule r_2 is defeated.

We notice that the preferences in the above examples, viewed under the revision program framework, are non-selecting preferences (Definition 4), and justified revisions are not models of the program. Theorem 3 discusses a condition under which $(P, \mathcal{L}, S)$-justified revisions are models of the original program P. We show next that under this condition and when only preferences with empty bodies are used, our framework coincides with preferred answer sets for prioritized logic programs [5].

Before we introduce the theorem about the relationship between revision program with preferences and preferred answer sets for prioritized logic programs, we need some more notation. First, we will assume that for every revision program with preferences $(P, \mathcal{L}, S)$, S is a set of selecting preferences, cycle-free, and the body in each preference of the form (5) in S is empty. We will refer to such programs as *static revision programs with preferences*. For such a program, we define a corresponding prioritized logic program $Q(P) = (lp(P), <)$ where $lp(P)$ is defined as in Section 2 and $<= \{(l_1, l_2) : prefer(l_1, l_2) \in S\}$.

Theorem 5. *Let $(P, \mathcal{L}, S)$ be a static revision program with preferences. Then, R is a $(P, \mathcal{L}, S)$-justified revision of the empty database iff R is a preferred answer set of $Q(P)$ as defined in [5].*

The proof of this property can be found in the appendix.

Our work in this paper is also strongly related to dynamic logic programming (DLP) [4]. DLP is introduced as a mean to update knowledge bases that might contain generalized logic programming rules. Roughly, a DLP is an ordered list of generalized logic programs, where each represents the properties of the knowledge base at a time moment. The semantics of a DLP – taking into consideration a sequence of programs up to a time point t – specifies which rules should be applied to derive the state of the knowledge base at t. It has been shown that DLP generalizes revision programming [4]. DLP has been extended to deal with preferences [3, 1]. A DLP with preferences, or a *prioritized DLP*, is a pair (P, R) of two DLPs; P is a labeled DLP whose language does not contain the binary predicate $<$ and R is a DLP whose language contains the binary predicate $<$ and whose set of constants includes all the rule labels from both programs. Intuitively, (P, R) represents a knowledge at different time moments – the same way a DLP does – with the exception that there are preferences between rules in (P, R). An atom of the form $r_1 < r_2$ represents the fact that rule r_1 is preferred to rule r_2. The semantics of prioritized DLP makes sure that the preference order between rules is reflected in the set of consequences derivable from the knowledge base. More precisely, for two conflicting rules r_1 and r_2, if $r_1 < r_2$ is derived, then the consequence of the rule r_1 should be preferred over the consequence of r_2. Prioritized DLP deals with preferences using the compilation approach. In fact, the approach coincides with that of preferred answer sets

for extended logic programs [5] when the DLP consists of a single program as shown in [3]. In this sense, the prioritized DLP approach is similar to the approach described in Section 3, in which we add to a revision program a preference relation between its rules and define the semantics of a revision program with preferences following the compilation approach. It follows from our discussion in the previous paragraph that revision programming with preferences and DLP with preferences will yield different results in certain situations. Other difference between our work and prioritized DLP lies in that we consider other types of preferences (e.g., maximal number of applicable rules, weighted rules, weighted atoms, or minimal size difference) and prioritized DLP does not. We plan to investigate the use of these types of preferences in DLP in the future.

Finally, DLP is also used as the main representation language for a multi-agent architecture in [12]. In this paper, we take the first step towards this direction by using revision programming with preferences to represent and reason about beliefs of multi-agents in a coordinated environment. A detailed comparison with MINERVA is planned in the near future.

6   Conclusions

The notion of preference has found pervasive applications in the context of knowledge representation and commonsense reasoning in MAS. Indeed a large number of approaches have been proposed to improve the knowledge representation capabilities of logic programming by introducing different forms of preferences. In this paper, we presented a novel extension of the revision programming framework which provides the foundations for expressing very general types of preferences. Preferences provide the ability to "defeat" the use of certain revision rules in the computation of the revisions; this allows us to either reduce the number of revisions generated (eventually leading to a single revision), or to generate revisions even in the presence of conflicting revision rules.

We proposed different preference schemes, starting from a relatively dynamic partial order between revision rules (*revision programs with preferences*), and then moving to a more general notion of weights, associated to revision rules and/or database atoms. Soft revision rules can be dynamically included or excluded from the generation of revisions depending on optimization criteria based on the weights of the revision (e.g., minimization of the total weight associated to the revision). We provided motivating examples for the different preference schemes, along with a precise description of how preferred revisions can be computed using the `smodels` answer set inference engine.

7   Appendix

In this section we give a proof of Theorem 5, that under certain conditions, the justified revisions of labeled revision programs with preferences coincide with the preferred answer sets of prioritized logic programs introduced in [5].

A prioritized logic program[2] is a pair $(P, <)$ where P is a logic program and $<$ is a preference relation among rules of P. The semantics of $(P, <)$ is defined by its *preferred answer set* - answer sets of P satisfying some conditions determined by $<$. We will first recall the notion of preferred answer sets from [5]. A binary relation R on a set S is called *strict partial order* (or *order*) if R is irreflexive and transitive. An order R is *total* if for every pair $a, b \in S$, either $(a, b) \in R$ or $(b, a) \in R$; R is *well-founded* if every set $X \subseteq S$ has a minimal element; R is *well-ordered* if it is total and well-founded.

Let P be a collection of rules of the form

$$r : \qquad l_0 \leftarrow l_1, \ldots, l_m, \; not \; l_{m+1}, \ldots, \; not \; l_n$$

where l_i's are ground literals. Literals $l_1, \ldots, l_m$ are called the *prerequisites* of r. If $m = 0$ then r is said to be *prerequisite free*. A rule r is *defeated* by a literal l if $l = l_i$ for some $i \in \{m + 1, \ldots, n\}$; r is defeated by a set of literals X if X contains a literal that defeats r. A program P is *prerequisite free* if every rule in P is prerequisite free. For a program P and a set of literals X, the *reduct of P with respect to X*, denoted by $^X P$, is the program obtained from P by

- deleting all rules with prerequisite l such that $l \notin X$; and
- deleting all prerequisites of the remaining rules.

Definition 11. *[5] Let $(P, <)$ be a prioritized logic program where P is prerequisite free and $<$ is a total order among rules of P. An answer set S of P is a preferred answer set of $(P, <)$ if $C_<(A) = A$ where (i) $C_<(A)$ is the smallest set of ground literals that is logically closed (wrt. P); (ii) $\bigcup_{i=0}^{\infty} S_i \subseteq C_<(A)$; and (iii) the sequence S_i is defined as follows:*

$$S_0 = \emptyset$$

$$S_n = \begin{cases} \bigcup_{i=0}^{n-1} S_i & \begin{array}{ll} if & r_n \text{ is defeated by } \bigcup_{i=0}^{n-1} S_i \\ or & r_n \text{ is defeated by } A \text{ and } head(r_n) \in A \end{array} \\ \\ \bigcup_{i=0}^{n-1} S_i \cup \{head(r_n)\} & otherwise \end{cases}$$

and r_n is the nth rule in the order $<$.

For an arbitrary prioritized logic program $(P, <)$, a set of literals A is called a preferred answer set of $(P, <)$ if it is a preferred answer set of $(^A P, <'))$ for some total order $<'$ that extends $^A <$ which inherits from $<$ by the map: $f : {}^A P \rightarrow P$, i.e., $r'_1 {}^A\!\!< r'_2$ if and only if $f(r'_1) < f(r'_2)$ where $f(r')$ is the first rule in P with respect to $<$ such that r' is obtained from r through the reduction ^{A}.

Now we are ready to give the proof of Theorem 5.

---

[2] In this appendix, by a logic program we mean a propositional logic program. This is because we only work with propositional revision programs.

Theorem 5. *Let $(P, \mathcal{L}, S)$ be a static revision program with preferences. Then, R is a $(P, \mathcal{L}, S)$-justified revision of the empty database if and only if R is a preferred answer set of $Q(P)$.*

Proof. Let U be the set of all atoms that appear in the program P.

($\Rightarrow$) Let R be a $(P, \mathcal{L}, S)$-justified revision of the empty database. We have that R is a model of P (Theorem 3). Hence, R satisfies the rules of $lp(P)$. Furthermore, there exist a $P^{S,\emptyset}$-justified revision R' such that $R' \cap U = R$.

We will first show that R is a minimal set of literals satisfying the rules of $lp(P)$. Assume the contrary, that there exists $M \subset R$ such that M satisfies the rules of $lp(P)$. Consider $a \in R \backslash M$. Since $a \in R$, there exists a rule r of $P^{S,\emptyset}$ such that $head(r) = \mathbf{in}(a)$, $\mathbf{in}(a) \in NC(P_{\emptyset, R'}^{S,\emptyset})$, and $body(r)$ is satisfied by R'. Because $r \in P$, we have that $ok(r) \in R'$. Hence, $lp(r) \in lp(P)$ and the body of r is satisfied by R. This contradicts the fact that M is closed under $lp(P)$. This allows us to conclude that R is an answer set of P.

It remains to be shown that R is a preferred answer set of $(lp(P), <)$. Consider the prioritized program $(^R(lp(P)), <')$ where $^R(lp(P))$ is the reduct of $lp(P)$ with respect to R and $<'$ inherits from $<$ (as defined in Definition 11). It follows from the definition of the reduct that if $r \in^R (lp(P))$ and r is not defeated by R then $head(r) \in R$.

We need to show that R is a preferred answer set of $(^R(lp(P)), <')$. Let $<^*$ be the transitive closure of $<'$, $RN = \{r \mid r \in (^R(lp(P)), \ r$ is not defeated by R and r does not occur in $<^*\}$, and $RD = \{r \mid r \in (^R(lp(P)), \ r$ is defeated by R and r does not occur in $<^*\}$. Let $rn_1, \ldots, rn_{n_1}$ be an enumeration of RN and $rd_1, \ldots, rd_{n_2}$ be an enumeration of RD. We define an ordering $<''$ on the rules of $(^R(lp(P)))$ as follows.

- $r <'' r'$ if $r <^* r'$;
- $rn_i <'' rn_j$ for $1 \leq i < j \leq n_1$;
- $rn_{n_1} <'' r$ for r occurs in $<^*$;
- $rd_i <'' rd_j$ for $1 \leq i < j \leq n_2$; and
- $r <'' rd_i$ for r occurs in $<^*$ and $1 \leq i \leq n_2$;

We have that $<''$ is a total order on the set of rules of $^R(lp(P))$. Let $r_1, \ldots, r_m$ be the sequence of rules of $^R(lp(P))$, ordered by $<''$. Let $S_0, \ldots, S_m$ be the sequence of sets of literals defined for $^R(lp(P))$ with respect to $<''$. It is easy to see that because R is an answer set of P, $\bigcup_{i=0}^{m} S_i \subseteq R$. Thus, we only need to show that for every $a \in R$, there exists $0 \leq j \leq m$ such that $a \in S_j$.

Consider an arbitrary $a \in R$. It follows from the definition of answer set that there exists some rule r of $lp(P)$ such that $head(r) = a$ and $body(r)$ is satisfied by R. This implies that the reduct r' of r belongs to $^R(lp(P))$. Clearly, r' is not defeated by R. Without the lost of generality, we can assume that $r' = r_l$ is the first rule in the sequence of the rules of $^R(lp(P))$ whose head is a. Together with the fact that $\bigcup_{i=0}^{l-1} S_i \subseteq R$, we can conclude that $head(r) \in S_l$. Thus, we have that $R \subseteq \bigcup_{i=0}^{m} S_i$. This, together with the fact that R is an answer set of $lp(P)$, shows that R is a preferred answer set of $(lp(P), <)$.

($\Leftarrow$) Let R be a preferred answer set of $Q(P)$, i.e., R is a preferred answer set of $(lp(P), <')$ for some total order $<'$ that extends $<$.

First, because R is an answer set of $lp(P)$ we have that R is a P-justified revision of $\emptyset$. Let $IN' = \mathit{Inertia}(\emptyset, R) = \{\mathbf{out}(a) : a \notin R\}$. We have that

$$R = \{a \in U : \mathbf{in}(a) \in NC(P_{\emptyset,R})\}$$

where $P_{\emptyset,R}$ consists of rules of the form

$$head(r) \leftarrow body(r) \setminus IN'$$

where $r \in P$ and where, by definition, $NC(P_{\emptyset,R})$ is the least model of $P_{\emptyset,R}$, when treated as a Horn program built of independent propositional atoms of the form $\mathbf{in}(a)$ and $\mathbf{out}(b)$. Let

$$
\begin{aligned}
d(R) &= \{\mathit{defeated}(r) \mid r \in P, \exists r'.[r' < r, R \text{ satisfies } body(r')]\}, \\
ok(R) &= \{ok(r) \mid r \in P, \mathit{defeated}(r) \notin d(R)\},
\end{aligned}
$$

and

$$R' = R \cup d(R) \cup ok(R) \cup S.$$

We will show that R' is a $P^{S,\emptyset}$-justified revision of $\emptyset$. Because the initial database is empty, we have that

$$\mathit{Inertia}(\emptyset, R') = \{\mathbf{out}(a) : a \in U^{\mathcal{L}(P)} \notin R'\}.$$

To simplify the presentation, let us denote $\mathit{Inertia}(\emptyset, R')$ by IN. From the construction of R', we have that $IN' = IN \cap \{\mathbf{out}(a) : a \in U\}$.

We will now construct the program $P' = P_{\emptyset,R'}^{S,\emptyset}$. We have that P' consists of the following rules:

(a) $head(r) \leftarrow body(r) \setminus IN, \mathbf{in}(ok(r))$ where r is a rule in P, $body(r) \setminus IN$ is the set of literals occurring in $body(r)$ which do not occur in IN.
(b) $\mathbf{in}(ok(r)) \leftarrow \mathbf{out}(\mathit{defeated}(r)) \setminus IN$;
(c) $\mathbf{in}(\mathit{prefer}(r, r')) \leftarrow$ if $\mathit{prefer}(r, r') \in S$;
(d) $\mathbf{in}(\mathit{defeated}(r)) \leftarrow body(r') \setminus IN, \mathbf{in}(\mathit{prefer}(r', r))$ if $\mathit{prefer}(r', r)) \in S$.

We will now show that R' is P'-justified revision of the empty database. It follows from Definition 1 that we need to show that $R' = \{a : \mathbf{in}(a) \in NC(P')\}$. Let $a \in R'$. We consider four cases:

- $a = ok(r)$ for some r. By construction of R', we have that $a \in R'$ iff $\mathit{defeated}(r) \notin d(R)$ iff $\mathit{defeated}(r) \notin R'$ iff $\mathbf{out}(\mathit{defeated}(r)) \in IN$ iff $\mathbf{in}(ok(r)) \in NC(P')$;
- $a = \mathit{prefer}(r, r')$. From the construction of R', $a \in R'$ iff $\mathit{prefer}(r, r') \in S$ iff $\mathbf{in}(\mathit{prefer}(r, r') \leftarrow$ belongs to P' iff $\mathbf{in}(\mathit{prefer}(r, r')) \in NC(P')$.
- $a \in U$. We will show that for every $a \in U$, $\mathbf{in}(a) \in NC(P')$ iff $a \in R'$ and $\mathbf{out}(a) \in NC(P')$ iff $a \notin R'$. Observe that for every rule of the type (a) we have that $head(r) \leftarrow body(r) \setminus IN'$ belongs to the program $P_{\emptyset,R}$. Therefore, $a \in R$ (resp. $a \notin R$) implies that $\mathbf{in}(a) \in NC(P_{\emptyset,R})$ (resp. $\mathbf{out}(a) \in NC(P_{\emptyset,R})$). Let T be the fix point operator that is used in computing the least fix point of the program $P_{\emptyset,R}$. We have that $a \in R$ (resp. $a \notin R$) if and only if there exists a minimal number k such that $\mathbf{in}(a) \in T^k(P_{\emptyset,R})$ and $\mathbf{in}(a) \notin T^i(P_{\emptyset,R})$ for $i < k$ (resp. $\mathbf{out}(a) \in T^k(P_{\emptyset,R})$ and $\mathbf{out}(a) \notin T^i(P_{\emptyset,R})$ for $i < k$). We can prove by induction over k that $\mathbf{in}(a) \in NC(P')$ (resp. $\mathbf{out}(a) \in NC(P')$):

- **Base:** $k = 0$ implies that $\mathbf{in}(a) = head(r)$ is a fact in $P_{\emptyset,R}$. Hence, $\mathbf{in}(a) \leftarrow \mathbf{in}(ok(r))$ is a rule in P'. We would like to show that $ok(r) \in R'$. Assume the contrary, $ok(r) \notin R'$. This implies that there exists a rule r' in P such that $prefer(r', r) \in S$ and R satisfies $body(r')$. Because $(P, \mathcal{L}, S)$ is static, we have that (i) $body(r') \cup body(r)$ is incoherent; or (ii) $(head(r))^D \in body(r')$ and $(head(r'))^D \in body(r)$. Since the body of r is empty, (i) cannot happen. If (ii) happens, we have that R cannot satisfy the body of r' due to the fact that R is a P-justified revision of $\emptyset$. This implies that our assumption is incorrect, i.e., $ok(r) \in R'$. From the first item, we have that $ok(r)$ is a fact in P'. Thus, $\mathbf{in}(a) \in NC(P')$. Similar argument allows us to conclude that if $\mathbf{out}(a) \in T^0(\emptyset)$ then $\mathbf{out}(a) \in NC(P')$.
- **Step:** Assume that we have proved the conclusion for k. We need to show that if $\mathbf{in}(a) \in T^{k+1}(P_{\emptyset,R})$, then $\mathbf{in}(a) \in NC(P')$. Similar to the base case, we can show that there exists a rule r or P such that $head(r) = \mathbf{in}(a)$, $body(r) \setminus IN \subseteq NC(P')$ and $\mathbf{in}(ok(r)) \in NC(P')$. This allows us to conclude that $\mathbf{in}(a) \in NC(P')$. The same argument holds for $\mathbf{out}(a) \in T^{k+1}(P_{\emptyset,R})$. This proves the inductive step.

- $a = defeated(r)$ for some r. Then, $a \in R'$ if and only if there exists a rule r', $prefer(r', r) \in S$ such that the body of r' is satisfied by R. Thus, $body(r') \setminus IN$ is satisfied by R', i.e., $\mathbf{in}(defeated(r)) \in NC(P')$.

The above items show that $a \in R'$ if and only if $\{a \mid \mathbf{in}(a) \in NC(P')\}$. This implies that R' is a $P^{S,\emptyset}$-justified revision of $\emptyset$, i.e. R is a $(P, \mathcal{L}, S)$-justified revision of $\emptyset$.   □

References

1. J.J. Alferes, P. Dell'Acqua, and L.M. Pereira. A compilation of updates plus references. In *Logics in Artificial Intelligence, European Conference*, pages 62–73. Springer, 2002.
2. J.J. Alferes, F. Banti, A. Brogi, J.A. Leite. Semantics for Dynamic Logic Programming: A Principle-Based Approach. In *LPNMR*, pages 8–20. Springer Verlag, 2004.
3. J.J. Alferes and L.M. Pereira. Updates plus preferences. In *Logics in Artificial Intelligence, European Workshop (JELIA)*, pages 345–360. Springer, 2000.
4. J.J. Alferes, J.A. Leite, L.M. Pereira, H. Przymusinska, and T.C. Przymusinski. Dynamic Updates of Non-monotonic Knowledge Bases. *JLP*, 45, 2000.
5. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1–2):297–356, 1999.
6. E. Delgrande, T. Schaub, and H. Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3(2):129–187, March 2003.
7. H.E. Durfee. *Coordination of Distributed Problem Solvers*. Kluwer Academic Press, 1988.
8. T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. Using Methods of Declarative Logic Programming for Intelligent Information Agents. *TPLP*, 2(6), 2002.
9. P. Faratin and B. Van de Walle. Agent Preference Relations: Strict, Indifferent, and Incomparable. In *AAMAS*. ACM, 2002.
10. M. Gelfond and T.C. Son. Prioritized default theory. In *Selected Papers from the Workshop on Logic Programming and Knowledge Representation 1997*, pages 164–223, Springer Verlag, LNAI 1471, 1998.
11. P. La Mura and Y. Shoham. Conditional, Hierarchical, Multi-agent Preferences. In *TARK VII*, 1998.

12. J.A. Leite, J.J. Alferes, and L.M. Pereira. MINERVA: a Dynamic Logic Programming Agent Architecture. In *Intelligent Agents VIII*, pages 141–157. Springer Verlag, 2002.
13. J. Liu and Y. Ye. *E-Commerce Agents*. Lecture Notes in AI, Springer Verlag, 2001.
14. W. Marek, I. Pivkina, and M. Truszczyński. Revision programming = logic programming + integrity constraints. In *Computer Science Logic*, Springer Verlag, 1999.
15. W. Marek and M. Truszczyński. Revision programming, database updates and integrity constraints. In *ICDT*, pages 368–382. Springer Verlag, 1995.
16. W. Marek and M. Truszczyński. Revision programming. *Theoretical Computer Science*, 190(2):241–277, 1998.
17. I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In *JICSLP*, pages 289–303. MIT Press, 1996.
18. H. Prakken. *Logical Tools for Modeling Legal Arguments*. Kluwer Publishers, 1997.
19. T. Przymusinski and H. Turner. Update by means of Inference rules. In *LPNMR*, pages 156–174. Springer Verlag, 1995.
20. C. Sakama and K. Inoue. Updating Extended Logic Programs through Abduction. In *LPNMR*, pages 147–161. Springer Verlag, 1999.
21. M. De Voss and D. Vermeir. A Logic for Modeling Decision Making with Dynamic Preferences, In *Logics in Artificial Intelligence, European Workshop (JELIA)*, pages 391–406. Springer, 2000.