# Declarative Semantics for Revision Programming and Connections to Active Integrity Constraints

Luciano Caroprese[1] and Mirosław Truszczyński[2]

[1] Università della Calabria, 87030 Rende, Italy, caroprese@deis.unical.it
[2] Department of Computer Science, University of Kentucky, Lexington, KY 40506, USA, mirek@cs.uky.edu

**Abstract.** We investigate *revision programming*, a formalism to describe constraints on belief sets (databases, knowledge bases), and to specify *preferred* ways to enforce them. We propose several semantics for revision programs combining ideas from logic programming and *active integrity constraints*, a formalism to model preferred ways to enforce integrity constraints on databases. We present results on the complexity of the semantics we introduce. We also show that all these semantics are invariant under "shifting." Finally, we prove that from the perspective of a broad semantic landscape of revision programming, there is a direct correspondence between revision programs and active integrity constraints.

## 1 Introduction

Revision programming [1, 2] was proposed as a formalism to describe revisions of sets (state descriptions, belief sets, databases, etc). To focus our attention, we will discuss revision programs here in the context of databases but, as just noted, their applicability goes beyond that setting.

Revision programs consist of *revision rules*. They specify conditions a revised database has to satisfy, as well as preferred ways to enforce them. To describe formally the meaning of normal revision programs, researchers proposed the semantics of *justified* and *supported* revisions [2], motivated by the stable-model semantics [3] and the supported-model semantics [4] of logic programs, respectively. The semantics of justified revisions was later generalized to the case of disjunctive revision programs in [5]. In this paper, for consistency with other notation, we refer to it as the semantics of justified *weak* revisions.

The definitions of the two semantics of revision programs do not directly take into account the postulate of the *minimality of change*, which prefers as the results of revision those databases that differ minimally from the original ones. Not surprisingly, the postulate holds neither for justified weak revisions, a shortcoming that has apparently been overlooked so far, nor for the semantics of supported revisions. Second, while in the restricted case of *normal* revision programs justified weak revisions are change-minimal, it is a side-effect of other aspects of the definition rather than a result of an explicit design decision. Since the minimality of change is among the most basic principles of database update, belief revision and nonmonotonic reasoning (cf. for instance, [6, 7]), it is important to study its effect on the semantics of revision programs. This is one of the objectives of this paper.

The theory of revision programs we develop here follows our recent investigations of *active integrity constraints* [8]. An *integrity constraint* is a condition on a database. If the database violates an integrity constraint, it needs to be *repaired* — updated so that the integrity constraint holds again. Often, there are several ways to do so. An *active* integrity constraint encodes *explicitly* both an integrity constraint and preferred basic actions to repair it, if it is violated. To specify the meaning of active integrity constraints, [9] proposed the notion of a *founded repair*. Founded repairs are change-minimal and satisfy a certain groundedness condition. In [8], we proposed several additional semantics for active integrity constraints. Together with founded repairs, they cover a spectrum of features one might require of database repairs. The class consists of the semantics of weak repairs, repairs, founded weak repairs, founded repairs, justified weak repairs, and justified repairs. The term *weak* points to the fact that the corresponding semantics is not required to have the minimality of change property. The terms *founded* and *justified* refer to different "grounding" principles used in defining the semantics.

We show[1] that this general schema for defining repairs can be lifted to revision programs, and yields the semantics of weak revisions, revisions, founded weak revisions, founded revisions, justified weak revisions, and justified revisions. We show that the semantics of founded weak revisions is an extension of the semantics of supported revisions to the case of disjunctive revision programs, and observe that the semantics presented in [5] appears in the schema under the name of justified *weak* revisions, as it does not satisfy the minimality-of-change property.

The relationship to active integrity constraints is not coincidental. Active integrity constraints, while different in several technical details, share with revision programs the same basic motivation of guiding the database update process through user-specified preferences. To make the connection between the two formalisms precise and explicit, we present a mapping from active integrity constraints to revision programs, under which the corresponding semantics on each side coincide.

The class of revision programs is in a certain sense broader than the class of active integrity constraints. There is a precise match between the two, however, if we limit attention to a subclass of revision programs that we refer to as *proper*. We prove that the restriction does not affect the expressive power of revision programming. Thus, our results show that both formalisms, even though syntactically different and originally endowed with different semantics, can be regarded as notational variants of each other.

One of the properties we establish for all semantics we discuss here is *invariance under shifting*. We use it to relate revision programs and logic programs of Lifschitz and Woo [10], aligning justified weak repairs with answer sets. We also point out that all other semantics of revision programs can be adapted for Lifschitz-Woo programs.

## 2 Revision Programming — an Overview

In this section we review the basic terminology of revision programming, and recall the two semantics introduced in [1, 2, 5]: the semantics of supported revisions, and the

[1] We omit proofs due to space restrictions. They can be found at `www.ca.uky.edu/ai/rp-full.pdf`.

semantics of justified weak revisions (originally referred to in [5] as justified revisions and renamed here for consistency with the general naming schema we use).

We consider a finite set $At$ of propositional atoms, representing all *ground* atoms in a language of predicate logic with a finite set of constants and with no function symbols. *Databases* are subsets of $At$. A *revision literal* is an expression $\mathbf{in}(a)$ or $\mathbf{out}(a)$, where $a \in At$. Revision literals $\mathbf{in}(a)$ and $\mathbf{out}(a)$ are *duals* of each other. If $\alpha$ is a revision literal, we denote its dual by $\alpha^D$. We extend this notation to sets of revision literals. We say that a set of revision literals is *consistent* if it does not contain a pair of dual literals. Revision literals represent elementary updates one can apply to a database. We define the result of applying a *consistent* set $\mathcal{U}$ of revision literals to a database $\mathcal{I}$ as follows:

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

A *revision rule* is an expression of the form

$$r = \alpha_1 | \ldots | \alpha_k \leftarrow \beta_1, \ldots, \beta_m, \tag{1}$$

where $k, m \geq 0$, $k+m \geq 1$, and $\alpha_i$ and $\beta_j$ are revision literals. The set $\{\alpha_1, \ldots, \alpha_k\}$ is the *head* of the rule (1); we denote it by $head(r)$. Similarly, the set $\{\beta_1, \ldots, \beta_m\}$ is the *body* of the rule (1); we denote it by $body(r)$. A revision rule is *normal* if $|head(r)| \leq 1$. A *revision program* is a collection of revision rules. A revision program is *normal* if all its rules are normal.

A database $\mathcal{D}$ *satisfies* a revision literal $\mathbf{in}(a)$ ($\mathbf{out}(b)$, respectively), if $a \in \mathcal{D}$ ($b \notin \mathcal{D}$, respectively). A database $\mathcal{D}$ *satisfies* a revision rule (1) if it satisfies at least one literal $\alpha_i$, $1 \leq i \leq k$, whenever it satisfies every literal $\beta_j$, $1 \leq j \leq m$. Finally, a database $\mathcal{D}$ satisfies a revision program $P$, if $\mathcal{D}$ satisfies every rule in $P$. We use the symbol $\models$ to denote the satisfaction relation.

For revision literals $\alpha = \mathbf{in}(a)$ and $\beta = \mathbf{out}(b)$, we set $lit(\alpha) = a$ and $lit(\beta) = not\, b$. We extend this notation to sets of revision literals. We note that every database interprets revision literals and the corresponding propositional literals in the same way. That is, for every database $\mathcal{I}$ and for every set of revision literals $L$, $\mathcal{I} \models L$ if and only if $\mathcal{I} \models lit(L)$.

It follows that a revision rule (1) specifies an integrity constraint equivalent to the propositional formula: $lit(\beta_1), \ldots, lit(\beta_m) \supset lit(\alpha_1), \ldots, lit(\alpha_k)$. However, a revision rule is not only an integrity constraint. Through its syntax, it also encodes a preference on how to "fix" a database, when it violates the constraint. Not satisfying a revision rule $r$ means satisfying all revision literals in the body of $r$ *and* not satisfying any of the revision literals in the head of $r$. Thus, enforcing the constraint means constructing a database that (1) does not satisfy some revision literal in the body of $r$, *or* (2) satisfies at least one revision literal in the head of $r$. The underlying idea of revision programming is to prefer those revisions that result in databases with the property (2).

As an example, let us consider the revision rule $r = \mathbf{in}(a) \leftarrow \mathbf{out}(b)$, and the empty database $\mathcal{I}$. Clearly, $\mathcal{I}$ does not satisfy $r$. Although $\mathcal{I}$ can be fixed either by inserting $a$, so that $head(r)$ becomes *true*, or by inserting $b$, so that $body(r)$ becomes *false*, the syntax of $r$ makes the former preferred.

Normal revision programs were introduced and studied in [1, 2] and the semantics of supported and justified weak revisions for normal revision programs were proposed

there. In [5], the formalism was extended to allow disjunctions of revision literals in the heads of rules, and the semantics of justified weak revisions was generalized to that case. We will now recall these definitions.

First, we define the notion of the *inertia set*. Let $\mathcal{I}$ and $\mathcal{R}$ be databases. We define the *inertia set* wrt $\mathcal{I}$ and $\mathcal{R}$, denoted $I(\mathcal{I}, \mathcal{R})$, by setting

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

In other words, $I(\mathcal{I}, \mathcal{R})$ is the set of all *no-effect* revision literals for $\mathcal{I}$ and $\mathcal{R}$, that is, revision literals that have no effect when revising $\mathcal{I}$ into $\mathcal{R}$.

Now, let $P$ be a *normal* revision program and $\mathcal{R}$ be a database. By $P_{\mathcal{R}}$ we denote the program obtained from $P$ by removing each rule $r \in P$ such that $\mathcal{R} \not\models \mathit{body}(r)$.

**Definition 1** (SUPPORTED UPDATES AND SUPPORTED REVISIONS). *Let $P$ be a normal revision program and $\mathcal{I}$ a database. A set $\mathcal{U}$ of revision literals is a* supported update *of $\mathcal{I}$ wrt $P$ if $\mathcal{U}$ is consistent and $\mathcal{U} = \mathit{head}(P_{\mathcal{I} \oplus \mathcal{U}})$. A set $\mathcal{E}$ is a* supported revision *of $\mathcal{I}$ wrt $P$ if $\mathcal{E} = \mathcal{U} \setminus I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$.* □

Intuitively, a consistent set $\mathcal{U}$ of revision literals is a supported update if it is precisely the set of literals "supported" by $P$ and the database resulting from updating $\mathcal{I}$ with $\mathcal{U}$. Eliminating from a supported revision all no-effect literals yields a supported revision.

While not evident explicitly from the definition, supported updates and revisions guarantee constraint enforcement (cf.[2]).

**Proposition 1.** *Let $P$ be a normal revision program and $\mathcal{I}$ a database. If $\mathcal{E}$ is a supported revision of $P$, then $\mathcal{I} \oplus \mathcal{E} \models P$.* □

Supported updates do not take into account the inertia set. Supported revisions do, but only superficially: simply removing no-effect literals from the corresponding supported update. Consequently, supported updates and revisions allow for a possibility for circularity of support and non-minimality.

*Example 1.* Let $P$ be a revision program containing the rules $\{\mathbf{in}(a) \leftarrow \mathbf{in}(b),\ \mathbf{in}(b) \leftarrow \mathbf{in}(a), \mathbf{in}(c) \leftarrow \mathbf{out}(d)\}$, and let $\mathcal{I}$ the empty database. $\mathcal{I}$ does not satisfy $P$ as it violates the rule $\mathbf{in}(c) \leftarrow \mathbf{out}(d)$. One can check that set $\mathcal{U} = \{\mathbf{in}(a), \mathbf{in}(b), \mathbf{in}(c)\}$ modeling the insertions of $a$, $b$ and $c$, is a supported update and a supported revision. However it is not minimal as its subset $\{\mathbf{in}(c)\}$ is sufficient to guarantee the satisfaction of $P$. □

The problem in the previous example is the circularity of support between $\mathbf{in}(a)$ and $\mathbf{in}(b)$. Each of them supports the other one but the set containing both is superfluous. To address the problem, [1, 2] proposed for normal revision programs the semantics of justified weak revisions, later extended to the disjunctive case in [5]. The idea was to "ground" justified weak revisions in the program and the inertia set by means of a *minimal closure*.

A set $\mathcal{U}$ of revision literals is *closed* under a revision program $P$ (not necessarily normal) if for every rule $r \in P$, whenever $\mathit{body}(r) \subseteq \mathcal{U}$, then $\mathit{head}(r) \cap \mathcal{U} \neq \emptyset$. If $\mathcal{U}$ is closed under $P$ and for every set $\mathcal{U}' \subseteq \mathcal{U}$ closed under $P$, we have $\mathcal{U}' = \mathcal{U}$, then $\mathcal{U}$ is a *minimal closed* set for $P$. Clearly, in general a revision program may admit more than one minimal closed set of revision literals.

**Definition 2.** [JUSTIFIED UPDATES AND JUSTIFIED WEAK REVISIONS] *Let $P$ be a revision program and let $\mathcal{I}$ be a database. A consistent set $\mathcal{U}$ of revision literals is a* $P$-justified update *for $\mathcal{I}$ if it is a minimal set closed under $P \cup I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$.*

*If $\mathcal{U}$ is a $P$-justified update for $\mathcal{I}$, then $\mathcal{U} \setminus I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$ is a* $P$-justified weak revision *for $\mathcal{I}$.* □

The inertia set plays an essential role in the definition, as it is used directly in the definition of a $P$-justified update. Again, it is not self-evident from the definition that justified updates and justified weak revisions, when applied to an initial database yield a database satisfying the program. However, the definition does indeed imply so [2, 5].

**Proposition 2.** *Let $P$ be a revision program and $\mathcal{I}$ a database. If $\mathcal{U}$ is a justified update or justified weak revision of $P$, then $\mathcal{I} \oplus \mathcal{U} \models P$.* □

## 3 A Family of Declarative Semantics for Revision Programming

The two semantics in the previous section were defined based on how revisions are "grounded" in a program, an initial database, and the inertia set. Two fundamental postulates of constraint enforcement and minimality of change played no explicit role in that research. The first one is no problem as it is a side effect of each of the two types of groundedness considered (cf. Propositions 1 and 2). The second one simply does not hold for supported revisions. And while [2] proved that justified weak revisions are change-minimal in the case of *normal* revision programs, it is not so in the general case.

*Example 2.* Let $P$ be a revision program consisting of the rules $\{\mathbf{in}(a)|\mathbf{out}(b) \leftarrow \mathbf{out}(c), \mathbf{out}(a)|\mathbf{in}(b) \leftarrow \mathbf{out}(c)\}$ and the $\mathcal{I}$ the empty database. It is easy to verify that set $\mathcal{U} = \{\mathbf{in}(a), \mathbf{in}(b)\}$ is a justified weak revision. However it is not minimal as $\mathcal{I}$ is already consistent and no update is needed. □

We will now develop a range of semantics for revision programs by taking the postulates of constraint enforcement and minimality of change explicitly into consideration.

**Definition 3.** [WEAK REVISIONS AND REVISIONS] *A set $\mathcal{U}$ of revision literals is a* weak revision *of $\mathcal{I}$ wrt a revision program $P$ if (1) $\mathcal{U} \cap I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U}) = \emptyset$ (relevance — all revision literals in $\mathcal{U}$ actually change $\mathcal{I}$ or, in other words, none of them is a no-effect literal wrt $\mathcal{I}$ and $\mathcal{I} \oplus \mathcal{U}$); and (2) $\mathcal{I} \oplus \mathcal{U} \models P$ (constraint enforcement). Further, $\mathcal{U}$ is a* revision *of $\mathcal{I}$ wrt a revision program $P$ if it is a weak revision and for every $\mathcal{U}' \subseteq \mathcal{U}$, $\mathcal{I} \oplus \mathcal{U}' \models P$ implies that $\mathcal{U}' = \mathcal{U}$ (minimality of change).* □

*Example 3.* Let $P$ be a revision program consisting of a rule $\mathbf{out}(b) \leftarrow \mathbf{in}(a)$, and let $\mathcal{I} = \{a, b\}$ be a database. Clearly, $\mathcal{I}$ does not satisfy $P$. The program $P$ has three weak revisions: $\mathcal{U}_1 = \{\mathbf{out}(a)\}$, $\mathcal{U}_2 = \{\mathbf{out}(b)\}$ and $\mathcal{U}_3 = \{\mathbf{out}(a), \mathbf{out}(b)\}$, respectively. The sets $\mathcal{U}_1$ and $\mathcal{U}_2$ are revisions. The set $\mathcal{U}_3$ is not. □

(Weak) revisions do not reflect the preferences on how to revise a database encoded in the syntax of revision rules. We will now introduce semantics that aim to capture that preference. First, we define a new semantics for revision programs by imposing change-minimality on justified weak revisions.

**Definition 4.** [JUSTIFIED REVISIONS] *Let $P$ be a revision program and let $\mathcal{I}$ be a database. A $P$-justified weak revision $\mathcal{E}$ for $\mathcal{I}$ is a $P$-*justified revision *for $\mathcal{I}$ if $\mathcal{E}$ is a revision of $\mathcal{I}$ wrt $P$ (that is, for every set $\mathcal{E}' \subseteq \mathcal{E}$ such that $\mathcal{I} \oplus \mathcal{E}' \models P$, $\mathcal{E}' = \mathcal{E}$).* □

Justified revisions have several useful properties. They are change-minimal and are grounded in the program *and* the inertia set. However, as stable models of logic programs, with which they share several similarities, justified revisions are not designed to handle reasoning by cases.

*Example 4.* Let $P = \{\mathbf{in}(b) \leftarrow \mathbf{in}(a),\ \mathbf{in}(b) \leftarrow \mathbf{out}(a),\ \mathbf{in}(a) \leftarrow \mathbf{in}(b)\}$ and let $\mathcal{I} = \emptyset$. A possible interpretation of the first two revision rules could be that no matter what the status of $a$ is, $b$ must be in the database. By the third rule $a$ must belong to the database, too. Thus, the program justifies the set $\mathcal{R} = \{\mathbf{in}(a), \mathbf{in}(b)\}$ as a revision of $\mathcal{I}$ (assuming that we allow reasoning by cases). It is easy to verify that $\mathcal{R} = \{\mathbf{in}(a), \mathbf{in}(b)\}$ is a revision of $\mathcal{I}$. However, it is not $P$-justified (weak) revision of $\mathcal{I}$. □

To provide a semantics capturing such justifications, we introduce now the concept of foundedness and the semantics of founded (weak) revisions,

**Definition 5.** [FOUNDED (WEAK) REVISIONS] *Let $\mathcal{I}$ be a database, $P$ a revision program and, and $\mathcal{E}$ a consistent set of revision literals.*

1. *A revision literal $\alpha$ is $P$-*founded *wrt $\mathcal{I}$ and $\mathcal{E}$ if there is $r \in P$ such that $\alpha \in head(r)$, $\mathcal{I} \oplus \mathcal{E} \models body(r)$, and $\mathcal{I} \oplus \mathcal{E} \models \beta^D$, for every $\beta \in head(r) \setminus \{\alpha\}$.*
2. *The set $\mathcal{E}$ is $P$-*founded *wrt $\mathcal{I}$ if every element of $\mathcal{E}$ is $P$-founded wrt $\mathcal{I}$ and $\mathcal{E}$.*
3. *$\mathcal{E}$ is a $P$-*founded (weak) revision *for $\mathcal{I}$ if $\mathcal{E}$ is a (weak) revision of $\mathcal{I}$ wrt $P$ and $\mathcal{E}$ is $P$-*founded *wrt $\mathcal{I}$.* □

One can verity that revision $\mathcal{R}$ in Example 4 is founded. We also note that condition (3) of the definition guarantees that founded (weak) revisions enforce constraints of the revision program. Next, directly from the definition, it follows that founded weak revisions are weak revisions. Similarly, founded revisions are revisions and so, they are change-minimal. Furthermore, founded revisions are founded weak revisions. However, there are (weak) revisions that are not founded, and founded weak revisions are not necessarily founded revisions, that is, are not change-minimal. The latter observation shows that foundedness is too weak a condition to guarantee change-minimality.

*Example 5.* Let $P$ be the revision program containing the rules $\{\mathbf{in}(b) \leftarrow \mathbf{in}(a),\ \mathbf{in}(a) \leftarrow \mathbf{in}(b), \mathbf{in}(c) \leftarrow \mathbf{out}(d)\}$ and $\mathcal{I}$ the empty database. The set $\{\mathbf{in}(d)\}$ is a revision of $\mathcal{I}$ wrt $P$. Therefore it is a weak revision of $\mathcal{I}$ wrt $P$. However, it is not a $P$-founded weak revision for $\mathcal{I}$. Therefore, it is not a $P$-founded revision for $\mathcal{I}$, either. The set $\{\mathbf{in}(c), \mathbf{in}(a), \mathbf{in}(b)\}$ is a $P$-founded weak revision for $\mathcal{I}$ but not a $P$-founded revision for $\mathcal{I}$. Indeed, $\{\mathbf{in}(c)\}$ is also a revision of $\mathcal{I}$ wrt $P$. □

In the case of normal revision programs, founded weak revisions coincide with supported revisions.

**Theorem 1.** *Let $P$ be a normal revision program and $\mathcal{I}$ a database. A set $\mathcal{E}$ of revision literals is a $P$-founded weak revision of $\mathcal{I}$ if and only if $\mathcal{E}$ is a $P$-supported revision of $\mathcal{I}$.* □

Foundedness is less restrictive than the condition defining justified updates, which is behind justified (weak) revisions.

**Theorem 2.** *Let $P$ be a revision program and let $\mathcal{I}$ be a database. If a set $\mathcal{E}$ of revision literals is a $P$-justified (weak) revision of $\mathcal{I}$, then it is a $P$-founded (weak) revision of $\mathcal{I}$.* □

The converse implications do not hold in general (cf. Example 4).

To summarize our discussion so far, revision programs can be assigned the semantics of (weak) revisions, justified (weak) revisions and founded (weak) revisions. Let us denote the classes of the corresponding types of revisions by $\mathbf{Rev}(\mathcal{I}, P)$, $\mathbf{WRev}(\mathcal{I}, P)$, $\mathbf{JRev}(\mathcal{I}, P)$, $\mathbf{JWRev}(\mathcal{I}, P)$, $\mathbf{FRev}(\mathcal{I}, P)$ and $\mathbf{FWRev}(\mathcal{I}, P)$. The containment relations are demonstrated in Figure 1. None of the containment relations can be replaced with the equality.

$$\begin{array}{ccccc} \mathbf{JRev}(\mathcal{I},P) & \subseteq & \mathbf{FRev}(\mathcal{I},P) & \subseteq & \mathbf{Rev}(\mathcal{I},P) \\ \cap\!| & & \cap\!| & & \cap\!| \\ \mathbf{JWRev}(\mathcal{I},P) & \subseteq & \mathbf{FWRev}(\mathcal{I},P) & \subseteq & \mathbf{WRev}(\mathcal{I},P) \end{array}$$

**Fig. 1.** The containment relations for the semantics for revision programs

## 4 Properties of the semantics

In this section, we will present several properties of the semantics we introduced here. Specifically, we exhibit two cases when justified weak revisions and justified revisions coincide, we present some complexity results and discuss the shifting property.

**Programs whose justified weak revisions are change-minimal.** In general, justified weak revisions are not change-minimal. However, for normal revision programs, justified weak revisions are change-minimal [2]. The change-minimality holds also in the following setting.

**Theorem 3.** *Let $\mathcal{I}$ be a database and $P$ a revision program such that for each revision literal $\alpha \in \bigcup_{r \in P} head(r)$, $\mathcal{I} \models lit(\alpha^D)$. If $\mathcal{E}$ is a $P$-justified weak revision of $\mathcal{I}$, then $\mathcal{E}$ is a $P$-justified revision of $\mathcal{I}$.* □

Theorem 3 concerns the case when each revision literal in the head of a revision rule, if applied, would change the status of the underlying atom in the database. For instance, if the initial database is empty and all revision literals prescribed by revision rules are positive (i.e. of the form $\mathbf{in}(a)$), then justified weak revisions are guaranteed to be minimal and so, are justified revisions.

**Computation and Complexity Results for Revision Programming.** In this section we present the complexity of the basic reasoning task associated with revision programs: deciding the existence of a (weak) revision of a particular type.

**Theorem 4.** *Let $\mathcal{I}$ be a database and $P$ a normal revision program. Then checking if there exists a $P$-justified revision ($P$-justified weak revision, respectively) for $\mathcal{I}$ is an NP-complete problem.* □

**Theorem 5.** *Let $\mathcal{I}$ be a database and $P$ a revision program. Then checking if there exists a $P$-justified revision ($P$-justified weak revision, respectively) for $\mathcal{I}$ is a $\Sigma_2^P$-complete problem.* □

**Theorem 6.** *Let $\mathcal{I}$ be a database and $P$ a revision program. Then checking if there exists a $P$-founded revision ($P$-founded weak revision, respectively) for $\mathcal{I}$ is a $\Sigma_2^P$-complete (NP-complete, respectively) problem.* □

These results show that the condition defining justified updates already makes the problem of the existence of a justified weak revision $\Sigma_2^P$-complete. Imposing, in addition, the minimality of change (considering justified revisions) does not increase the complexity. Foundedness is an "easier" condition. Deciding the existence of a founded weak revision is NP-complete. In this setting, imposing the minimality of change (switching to founded revisions) makes a difference. The complexity grows to $\Sigma_2^P$-complete.

**Shifting theorem for revision programs.** We will now study invariance under shifting [2]. Shifting consists of transforming an instance $\langle \mathcal{I}, P\rangle$ of the database repair problem to an isomorphic instance $\langle \mathcal{I}', P'\rangle$ by "shifting" $\mathcal{I}$ to $\mathcal{I}'$ and changing $P$ to $P'$ to reflect the "shift" of the database. A semantics for database revision has the *shifting property* if the revisions of the "shifted" instance $\langle \mathcal{I}', P'\rangle$ are precisely the results of modifying the revisions of the original instance $\langle \mathcal{I}, P\rangle$ according to the shift $\mathcal{I} \rightarrow \mathcal{I}'$. The shifting property is important. If it holds for a semantics, the study of that semantics can be reduced to the case when the input database is empty. Often, it allows us to relate revision programming and logic programming with negation.

*Example 6.* Let $\mathcal{I} = \{a, b\}$ and let $P = \{\mathbf{out}(a)|\mathbf{out}(b) \leftarrow\}$. There are two justified (and so, also founded) revisions for $\langle \mathcal{I}, P\rangle$: $\mathcal{E}_1 = \{\mathbf{out}(a)\}$ and $\mathcal{E}_2 = \{\mathbf{out}(b)\}$. Let $\mathcal{W} = \{a\}$. To "shift" the instance $\langle \mathcal{I}, P\rangle$ wrt $\mathcal{W}$, we first modify $\mathcal{I}$ by changing the status in $\mathcal{I}$ of elements in $\mathcal{W}$, in our case, of $a$. Since $a \in \mathcal{I}$, we remove it. Thus, $\mathcal{I}$ "shifted" wrt $\mathcal{W}$ becomes $\mathcal{J} = \{b\}$. Next, we modify $P$ correspondingly, replacing revision literals involving $a$ by their duals. That results in $P' = \{\mathbf{in}(a)|\mathbf{out}(b) \leftarrow\}$. The resulting instance $\langle \mathcal{J}, P'\rangle$ has two founded/justified revisions: $\{\mathbf{in}(a)\}$ and $\{\mathbf{out}(b)\}$. They can be obtained from the founded/justified revisions for $\langle \mathcal{I}, P\rangle$ by replacing $\mathbf{out}(a)$ with $\mathbf{in}(a)$ and $\mathbf{in}(a)$ with $\mathbf{out}(a)$ (the latter does not apply in this example). In other words, the original update problem and its shifted version are isomorphic. □

The situation presented in Example 6 is not coincidental. In this section we present results showing that the semantics of (weak) revisions, founded (weak) revisions and justified (weak) revisions satisfy the shifting property. We start by observing that *shifting* a database $\mathcal{I}$ to a database $\mathcal{I}'$ can be modeled by means of the symmetric difference operator. Namely, we have $\mathcal{I}' = \mathcal{I} \div \mathcal{W}$, where $\mathcal{W} = \mathcal{I} \div \mathcal{I}'$. This identity shows that one can shift any database $\mathcal{I}$ into any database $\mathcal{I}'$ by forming a symmetric difference of $\mathcal{I}$ with some set of atoms $\mathcal{W}$ (specifically, $\mathcal{W} = \mathcal{I} \div \mathcal{I}'$). We will now extend the operation of shifting a database wrt $\mathcal{W}$ to the case of revision literals and (resp. revision rules and revision programs). To this end, we introduce a *shifting* operator $T_{\mathcal{W}}$.

**Definition 6.** *Let $\mathcal{W}$ be a database and $\ell$ a revision literal. We define*

$$T_{\mathcal{W}}(\ell) = \begin{cases} \ell^D & \text{if the atom of } \ell \text{ is in } \mathcal{W} \\ \ell & \text{if the atom of } \ell \text{ is not in } \mathcal{W} \end{cases}$$

*and we extend this definition to sets of revision literals. Furthermore, if op is an operator on sets of revision literals (such as conjunction or disjunction), for every set $X$ of revision literals we define $T_{\mathcal{W}}(op(X)) = op(T_{\mathcal{W}}(X))$. Finally, for a revision rule $r = \alpha_1 | \ldots | \alpha_k \leftarrow \beta_1, \ldots, \beta_m$, we set $T_{\mathcal{W}}(r) = T_{\mathcal{W}}(\alpha_1 | \ldots | \alpha_k) \leftarrow T_{\mathcal{W}}(\beta_1, \ldots, \beta_m)$, and for a revision program $P$, $T_{\mathcal{W}}(P) = \{T_{\mathcal{W}}(r) \mid r \in P\}$.*[2] □

**Theorem 7.** (SHIFTING THEOREM FOR REVISION PROGRAMS) *Let $\mathcal{I}$ and $\mathcal{W}$ be databases. For every revision program $G$ and every consistent set $\mathcal{E}$ of revision literals:*

1. *$\mathcal{E}$ is a (weak) revision for $\mathcal{I}$ wrt $G$ if and only if $T_{\mathcal{W}}(\mathcal{E})$ is a (weak) revision for $\mathcal{I}$ wrt $T_{\mathcal{W}}(G)$*
2. *$\mathcal{E}$ is a $G$-justified (weak) revision for $\mathcal{I}$ if and only if $T_{\mathcal{W}}(\mathcal{E})$ is a $T_{\mathcal{W}}(G)$-justified (weak) revision for $\mathcal{I}$*
3. *$\mathcal{E}$ is a $G$-founded (weak) revision for $\mathcal{I}$ if and only if $T_{\mathcal{W}}(\mathcal{E})$ is a $T_{\mathcal{W}}(G)$-founded (weak) revision for $\mathcal{I}$*

## 5 Connections between Revision Programs and Active Integrity Constraints

We will now relate revision programs to active integrity constraints [9], an earlier formalism for expressing integrity constraints and preferred ways to enforce them.

**Active integrity constraints — an overview.** An *update action* is an expression $+a$ or $-a$, where $a \in At$. It states that $a$ is to be inserted or deleted, respectively. A set $\mathcal{U}$ of update actions is *consistent* if it does not contain update actions $+a$ and $-a$, for any $a \in At$. For a database $\mathcal{D}$ and a consistent set $\mathcal{U}$ of update actions, we define the result of *updating $\mathcal{D}$ by means of $\mathcal{U}$* as the database

$$\mathcal{D} \circ \mathcal{U} = (\mathcal{D} \cup \{a \mid +a \in \mathcal{U}\}) \setminus \{a \mid -a \in \mathcal{U}\}.$$

An *integrity constraint* is a formula $r = L_1, \ldots, L_m \supset \bot$, where $L_i$, $1 \leq i \leq m$, are propositional literals (atoms $a$ and their negations $not\ a$, for $a \in At$), and ',' stands for the conjunction. A database $\mathcal{D}$ *satisfies* an integrity constraint $r$, if $\mathcal{D}$ – viewed as a propositional interpretation – satisfies $r$. Given a set $\eta$ of integrity constraints and a database $\mathcal{I}$, the problem of *database repair* is to update $\mathcal{I}$ with a set of update actions so that the resulting database satisfies all integrity constraints in $\eta$.

**Definition 7.** [WEAK REPAIRS AND REPAIRS] *Let $\mathcal{I}$ be a database and $\eta$ a set of integrity constraints. A* weak repair *for $\langle \mathcal{I}, \eta \rangle$ is a consistent set $\mathcal{U}$ of update actions such that $(\{+a \mid a \in \mathcal{I}\} \cup \{-a \mid a \in At \setminus \mathcal{I}\}) \cap \mathcal{U} = \emptyset$ ("essential" update actions only), and $\mathcal{I} \circ \mathcal{U} \models \eta$ (constraint enforcement).*

*A consistent set $\mathcal{U}$ of update actions is a* repair *for $\langle \mathcal{I}, \eta \rangle$ if it is a weak repair for $\langle \mathcal{I}, \eta \rangle$ and for every $\mathcal{U}' \subseteq \mathcal{U}$ such that $\mathcal{I} \circ \mathcal{U}' \models \eta$, $\mathcal{U}' = \mathcal{U}$ (minimality of change).* □

[2] We note that we overload the notation $T_{\mathcal{W}}$ and interpret it based on the type of the argument.

Let $r = a, b \supset \bot$, and let $\mathcal{I} = \{a, b\}$ be a database. Clearly, $\mathcal{I} \not\models r$. There are three possible weak repairs of $\mathcal{I}$ wrt $r$: $\{-a\}$, $\{-b\}$ and $\{-a, -b\}$. The first two are minimal and so, they are repairs. No weak repair and no repair is distinguished as preferred.

To model preferences on (weak) repairs, [9] modified the syntax of integrity constraints by allowing the user to list preferred update actions. We will now present that approach. For a propositional literal $L$, we write $L^D$ for the dual literal to $L$. Further, for a literal $L = a$ ($L = not\, a$), we define $ua(L) = +a$ ($ua(L) = -a$). Similarly, for an update action $\alpha = +a$ ($\alpha = -a$), we define $lit(\alpha) = a$ ($lit(\alpha) = not\, a$). We call $+a$ and $-a$ the *duals* of each other, and write $\alpha^D$ to denote the dual of an update action $\alpha$. We extend the notation introduced here to sets of literals and update actions. We define an *active integrity constraint* to be an expression of the form

$$r = L_1,\ \ldots,\ L_m \supset \alpha_1 | \ldots | \alpha_k \tag{2}$$

where $m, k \geq 0$, $m + k \geq 1$, $L_i$ are literals, $\alpha_j$ are update actions, and

$$\{lit(\alpha_1)^D, \ldots, lit(\alpha_k)^D\} \subseteq \{L_1, \ldots, L_m\} \tag{3}$$

The set $\{L_1, \ldots, L_m\}$ is the *body* of $r$; we denote it by $body(r)$. Similarly, the set $\{\alpha_1, \ldots, \alpha_k\}$ is the *head* of $r$; we denote it by $head(r)$.

An active integrity constraint $L_1,\ \ldots,\ L_m \supset \alpha_1 | \ldots | \alpha_k$ represents the integrity constraint $L_1,\ \ldots,\ L_m \supset \bot$. Thus, we say that a database $\mathcal{D}$ *satisfies* an active integrity constraint $r$ ($\mathcal{D} \models r$) if $\mathcal{D}$ satisfies the corresponding integrity constraint. In this way, the semantics of weak repairs and repairs lift to active integrity constraints. However, an active integrity constraint is more than an integrity constraint. It also specifies preferred update actions to use by listing them in the head.

The condition (3) ensures that an active integrity constraint supports only those update actions that pertain to its body and can fix it. It can be restated concisely as $[lit(head(r))]^D \subseteq body(r)$. We call literals in $[lit(head(r))]^D$ *updatable* by $r$, as they can be affected by an update action in $head(r)$. All other literals in $body(r)$ are *non-updatable* by $r$. We write $up(r)$ and $nup(r)$ for the sets of literals updatable and non-updatable by $r$, respectively.

Let $r = a, b \supset -b$ (cf. the previous example), and let $\mathcal{I} = \{a, b\}$ be a database. As before, $\mathcal{I} \not\models r$, and there are two repairs of $\mathcal{I}$ wrt $r$: $\{-a\}$ and $\{-b\}$. Now, since $r$ lists $-b$ as an update action to execute, the latter one is preferred. The semantics of (weak) repairs are insensitive to such preferences. To reflect them, [9] defined *founded repairs*. A related concept of a founded weak repair was introduced in [8].

**Definition 8.** [FOUNDED (WEAK) REPAIRS] *Let $\mathcal{I}$ be a database, $\eta$ a set of active integrity constraints, and $\mathcal{U}$ a consistent set of update actions.*

1. *An update action $\alpha$ is* founded *wrt $\langle \mathcal{I}, \eta \rangle$ and $\mathcal{U}$ if there is $r \in \eta$ such that $\alpha \in head(r)$, $\mathcal{I} \circ \mathcal{U} \models nup(r)$, and $\mathcal{I} \circ \mathcal{U} \models \beta^D$, for every $\beta \in head(r) \setminus \{\alpha\}$.*
2. *The set $\mathcal{U}$ is* founded *wrt $\langle \mathcal{I}, \eta \rangle$ if every element of $\mathcal{U}$ is founded wrt $\langle \mathcal{I}, \eta \rangle$ and $\mathcal{U}$.*
3. *$\mathcal{U}$ is a* founded (weak) repair *for $\langle \mathcal{I}, \eta \rangle$ if $\mathcal{U}$ is a (weak) repair for $\langle \mathcal{I}, \eta \rangle$ and $\mathcal{U}$ is* founded *wrt $\langle \mathcal{I}, \eta \rangle$.* □

Foundedness captures a certain notion of "groundedness". Let us assume that $r \in \eta$ and $I \not\models r$. If $\alpha$ is founded wrt $\langle \mathcal{I}, \eta \rangle$ and $\mathcal{U}$ by means of $r$, then all literals in $body(r)$ other than $lit(\alpha^D)$ are satisfied by $\mathcal{I} \circ \mathcal{U}$. Thus, if $\mathcal{U}$ is to enforce $r$, it must contain $\alpha$.

Founded repairs for $\langle \mathcal{I}, \eta \rangle$, despite being change-minimal and founded in $\langle \mathcal{I}, \eta \rangle$ ("grounded" in $\langle \mathcal{I}, \eta \rangle$) may still be self-justified. To address the problem [8] introduced justified (weak) repairs. To present the definition we need more terminology.

A set $\mathcal{U}$ of update actions is *closed* under an active integrity constraint $r$ if $nup(r) \not\subseteq lit(\mathcal{U})$, or $head(r) \cap \mathcal{U} \neq \emptyset$. A set $\mathcal{U}$ of update actions is *closed* under a set $\eta$ of active integrity constraints if it is closed under every $r \in \eta$. A *minimal* set closed under $\eta$ can be viewed as "forced" by $\eta$, as all its elements are necessary (no nonempty subset can be dropped without violating the closedness condition).

Another concept we need is that of *no-effect actions*. Let $\mathcal{I}$ and $\mathcal{R}$ be databases. An update action $+a$ (respectively, $-a$) is a *no-effect* action for $(\mathcal{I}, \mathcal{R})$ if $a \in \mathcal{I} \cap \mathcal{R}$ (respectively, $a \notin \mathcal{I} \cup \mathcal{R}$). That is, a no-effect action does not change the status of its underlying atom. We denote by $ne(\mathcal{I}, \mathcal{R})$ the set of all no-effect actions wrt $(\mathcal{I}, \mathcal{R})$.

**Definition 9.** [JUSTIFIED (WEAK) REPAIR] *Let $\mathcal{I}$ be a database, $\eta$ a set of active integrity constraints, and $\mathcal{U}$ a consistent set of update actions.*

1. *$\mathcal{U}$ is a justified action set for $\langle \mathcal{I}, \eta \rangle$ if $\mathcal{U}$ is a minimal set of update actions containing $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ and closed under $\eta$.*
2. *If $\mathcal{U}$ is a justified action set for $\langle \mathcal{I}, \eta \rangle$, then $\mathcal{E} = \mathcal{U} \setminus ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ is a* justified weak repair *for $\langle \mathcal{I}, \eta \rangle$. If in addition, for every $\mathcal{E}' \subseteq \mathcal{E}$ such that $\mathcal{I} \circ \mathcal{E}' \models \eta$, $\mathcal{E}' = \mathcal{E}$, then $\mathcal{E}$ is a* justified repair *for $\langle \mathcal{I}, \eta \rangle$.* □

Clearly, (founded, justified) weak repairs are (founded, justified) repairs. One can also show that justified (weak) repairs are founded (weak) repairs which, in turn, are (weak) repairs. There are no other inclusions between the six classes of weak repairs that we discussed here (cf. [8, 9] for details).

**Proper revision programs and a connection to active integrity constraints.** There are striking similarities between the formalisms of revision programs and active integrity constraints. However, to relate the two, we first need to restrict the syntax of revision programs. We then show that the restriction does not change their expressivity.

A *proper revision rule* is a revision rule that satisfies the following condition: the literal in the head of the rule is not the dual of any literal in the body of the rule. A revision program is *proper* if all its revision rules are proper.

**Theorem 8.** *Let $P$ be a revision program. There is a proper revision program $P'$ such that for every database $\mathcal{I}$, (weak) revisions of $\mathcal{I}$ wrt $P$ ($P$-founded (weak) revisions, $P$-justified (weak) revisions of $\mathcal{I}$, respectively) coincide with (weak) revisions of $\mathcal{I}$ wrt $P'$ ($P'$-founded (weak) revisions, $P'$-justified (weak) revisions of $\mathcal{I}$, respectively).*

Given a proper revision rule $r$ of the form

$$\alpha_1 | \ldots | \alpha_k \leftarrow \beta_1, \ldots \beta_m$$

we denote by $AIC(r)$ the active integrity constraint

$$lit(\beta_1), \ldots, lit(\beta_m), lit(\alpha_1)^D, \ldots, lit(\alpha_k)^D \supset ua(\alpha_1) | \ldots | ua(\alpha_k).$$

We note that if $r$ is a revision constraint ($k = 0$), $AIC(r)$ is simply an integrity constraint. The operator $AIC(\cdot)$ is extended to proper revision programs in the standard way. It is easy to show that for each database $\mathcal{D}$, $\mathcal{D} \models P$ if and only if $\mathcal{D} \models AIC(P)$. We now have the following result.

**Theorem 9.** *Let $P$ be a proper revision program. A set $\mathcal{E}$ of revision literals is a (weak) revision (resp. $P$-justified (weak) revision, $P$-founded (weak) revision) of $\mathcal{I}$ wrt $P$ if and only if $ua(\mathcal{E})$ is a (weak) repair (resp. justified (weak) repair, founded (weak) repair) for $\langle \mathcal{I}, AIC(P) \rangle$.*

The mapping $AIC(\cdot)$ is a bijection between proper revision programs and sets of active integrity constraints. Thus, it establishes an *exact* bidirectional match between the two formalisms (for all semantics considered). We note that the restriction to proper programs is essential as $AIC(\cdot)$, when used with all revision programs, is no longer one-to-one and, in some cases, maps semantically different rules onto the same active integrity constraint. For instance, programs consisting of $\mathbf{in}(a) \leftarrow \mathbf{out}(a)$ and $\mathbf{in}(a) \leftarrow$, respectively, behave differently wrt $\mathcal{I} = \emptyset$ (the first program does not define any justified revisions, the second one does: $\mathcal{E} = \{\mathbf{in}(a)\}$). Yet, both rules are mapped by $AIC(\cdot)$ onto $not\, a \rightarrow +a$.

## 6 Discussion and Conclusions

We studied a formalisms of revision programming designed to support modeling of constraints on databases (belief sets), as well as preferred ways to enforce them if they are violated. Revision programming was proposed in [1, 2] and further developed in [5]. However, the earlier work focused only on one semantics, the semantics of $P$-justified weak revisions. It is a limitation of the earlier work as, on the one hand, $P$-justified weak revisions do not satisfy the minimality of change property and, on the other, they may be too restrictive in situations when reasoning by cases may be justified.

Therefore, we proposed here several new declarative semantics for revision programs, by imposing the minimality of change property on $P$-justified revisions and/or by modifying the groundedness condition behind justified weak revisions. We studied properties of the resulting semantics. In particular, we established the complexity of several decision problems, and identified two classes of revision programs, for which justified weak revisions are change-minimal. Revision programming shows several similarities with the formalism of active integrity constraints [9]. We proposed an interpretation of revision rules as active integrity constraints and proved that under that interpretation, revision programming (restricted to proper revision programs) and the formalism of active integrity constrains are notational variants of each other.

Finally, we also proved that all semantics we studied satisfy the shifting property. We will now briefly point out that thanks to shifting, one can relate revision programs to Lifschitz-Woo programs [10], which generalize disjunctive logic programs by allowing the default negation in the heads of rules. Namely, *Lifschitz-Woo* programs consist of rules of the form

$$a_1 | \ldots | a_k | not\, b_1 | \ldots | not\, b_m \leftarrow c_1, \ldots, c_s, not\, d_1, \ldots not\, d_n \qquad (4)$$

where each $a_i, b_i, c_i$ and $d_i$ is an atom and $not$ is a default negation. Given a revision rule

$$\mathbf{in}(a_1)|\ldots|\mathbf{in}(a_k)|\mathbf{out}(b_1)|\ldots|\mathbf{out}(b_m) \leftarrow \mathbf{in}(c_1),\ldots,\mathbf{in}(c_s),\mathbf{out}(d_1),\ldots\mathbf{out}(d_n) \tag{5}$$

there is a clear correspondence between the two. Let us denote by $LW(\cdot)$ a mapping that assigns a Lifschitz-Woo rule (4) to a revision rule (5). The following result was obtained in [5].

**Theorem 10.** *Let $P$ be a revision program. Then, a set $\mathcal{U}$ of revision literals is a $P$-justified revision of $\emptyset$ if and only if $\mathcal{U} = \{\mathbf{in}(a) \mid a \in M\}$, where $M \subseteq At$ is a stable model of $LW(P)$ (according the the definition from [10]).*

Thanks to shifting, this result allows us to represent any revision program $P$ and a database $\mathcal{I}$ as a Lifschitz-Woo program so that justified revisions correspond precisely to stable models, a property also observed in [5]. However, we proved here that shifting holds for other semantics of revision programs, too. Thus, Theorem 10 also suggests that *all* these semantics could be adapted directly to the setting of Lifschitz-Woo programs and, subsequently, to the formalism of programs with nested expressions [11] (subsuming Lifschitz-Woo programs). It follows that these generalized variants of logic programming can be endowed with a richer family of semantics that goes beyond the basic one given by stable models.

## References


1. Marek, W., Truszczyński, M.: Revision specifications by means of programs. Proceedings of JELIA 1994. Volume 838 of LNCS., Springer (1994) 122–136
2. Marek, W., Truszczyński, M.: Revision programming. Theoretical Computer Science **190** (1998) 241–277
3. Gelfond, M., Lifschitz, V.: The stable semantics for logic programs. Proceedings of ICLP 1988, MIT Press (1988) 1070–1080
4. Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and data bases. Plenum Press, New York-London (1978) 293–322
5. Pivkina, I.: Revision programming: a knowledge representation formalism. PhD thesis, Department of Computer Science, University of Kentucky (2001) http://lib.uky.edu/ETD/ukycosc2001d00022/pivkina.pdf.
6. McCarthy, J.: Circumscription — a form of non-monotonic reasoning. Artificial Intelligence **13** (1980) 27–39
7. Winslett, M.: Updating Logical Databases. Cambridge University Press (1990)
8. Caroprese, L., Truszczyński, M.: Declarative semantics for active integrity constraints (2008) Submitted, available at http://www.cs.uky.edu/ai/aic-full.pdf.
9. Caroprese, L., Greco, S., Sirangelo, C., Zumpano, E.: Declarative semantics of production rules for integrity maintenance. Proceedings of ICLP 2006. Volume 4079 of LNCS., Springer (2006) 26–40
10. Lifschitz, V., Woo, T.: Answer sets in general nonmonotonic reasoning. Proceedings of KR 1992. Morgan Kaufmann (1992) 603–614
11. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Annals of Mathematics and Artificial Intelligence (1999) 369–389