Revision programming

Victor W. Marek
Mirosław Truszczyński
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0027

Abstract

In this paper we introduce revision programming — a logic-based framework for describing constraints on databases and providing a computational mechanism to enforce them. Revision programming captures those constraints that can be stated in terms of the membership (presence or absence) of items (records) in a database. Each such constraint is represented by a *revision rule* $\alpha \leftarrow \alpha_1, \ldots, \alpha_k$, where α and all α_i are of the form $\mathbf{in}(a)$ and $\mathbf{out}(b)$. Collections of revision rules form *revision programs*. Similarly as logic programs, revision programs admit both declarative and imperative (procedural) interpretations. In our paper, we introduce a semantics that reflects both interpretations. Given a revision program, this semantics assigns to any database $\mathcal{B}$ a collection (possibly empty) of *P-justified revisions* of $\mathcal{B}$. The paper contains a thorough study of revision programming. We exhibit several fundamental properties of revision programming. We study the relationship of revision programming to logic programming. We investigate complexity of reasoning with revision programs as well as algorithms to compute P-justified revisions. Most importantly from the practical database perspective, we identify two classes of revision programs, *safe* and *stratified*, with a desirable property that they determine for each initial database a unique revision.

1 Introduction

In this paper we propose a framework for studying the process of database revision. Revisions that we have in mind are specified by means of *revision programs* — sets of *revision rules* expressing constraints on presence or absence of data items (records) in databases. The rules or constraints are of two forms:

$$\mathbf{in}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n) \tag{1}$$

and

$$\mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n). \tag{2}$$

Such rules have several possible interpretations. For instance, under a *declarative* interpretation, the meaning of rule (1) is that

1. a belongs to the database $\mathcal{B}$ under consideration, or
2. $a_k \notin \mathcal{B}$, for some k, $1 \leq k \leq m$, or
3. $b_k \in \mathcal{B}$, for some k, $1 \leq k \leq n$.

A similar *declarative* interpretation can be offered for the rule of type (2).

But there is also an *imperative*, or computational, interpretation of rules (1) and (2). Namely, assume that all data items a_k, $1 \leq k \leq m$, belong to the current database and none of the data items b_l, $1 \leq l \leq n$ belongs to the current database. Then, in the case of rule (1), the item a should be added to the database (if it is not there already), and in the case of rule (2), a should be eliminated from the database (if it is there).

This simultaneously declarative and imperative character of revision rules makes the assignment of semantics to revision programs quite difficult. The imperative interpretation implies that the rules of the program should be used in the process of computing a revision of a database. The declarative interpretation requires that, after we terminate the computation of a revision, the revised database should satisfy the constraints specified by the program.

This declarative/imperative nature of revision rules is not unique to revision programming. For instance, logic programs [Llo84, Apt90] can be assigned similar interpretations. Namely, the clauses of the program can be regarded as describing constraints that need to be *satisfied* and, in the same time, as a computational tool needed to compute appropriate models. For Horn programs, the computation uses clauses as inference rules. That is, when the premises of a clause have all been already computed, the head of the clause becomes computed, too [vEK76]. Similar concepts of computation are available for DATALOG programs [Ull88]. In the case of the stable semantics of logic programs [GL88] the computational mechanism is that of default logic [Rei80], see [BF91, MT89] for more details.

In this paper, we propose a semantics for revision programs, called the *justified revision* semantics. This semantics is motivated by the stable model semantics of logic

programs and provides a computational mechanism that uses a revision program to produce revisions of input databases. In addition, once the revisions are computed, they satisfy all the constraints described by the program.

We will now briefly describe the process of computing a revised database $\mathcal{R}$ from an initial database $\mathcal{I}$ using the rules of a revision program P. The computation involves a fixpoint construction. That is, a candidate for a revised database is first proposed. Next, a decision is made whether the transition from $\mathcal{I}$ to $\mathcal{R}$ is justified under the program P. If so, $\mathcal{R}$ is regarded as a revision of $\mathcal{I}$ (and called a *P-justified revision* of $\mathcal{I}$). Otherwise, another candidate for a revised database is considered.

The key question is: how to decide whether a transition from $\mathcal{I}$ to $\mathcal{R}$ is justified under P. To answer it, observe that to every candidate $\mathcal{R}$ for a P-justified revision of $\mathcal{I}$ we can assign the set of these elements which *do not change status* as we pass from $\mathcal{I}$ to $\mathcal{R}$. This set is, of course, given by the complement of the symmetric difference of $\mathcal{I}$ and $\mathcal{R}$. Assuming that U is the set of all data items under consideration, this set is $(\mathcal{I} \cap \mathcal{R}) \cup (U \setminus (\mathcal{I} \cup \mathcal{R}))$. Observe now that there is an important distinction between the elements of $(\mathcal{I} \cap \mathcal{R})$ and the elements of $U \setminus (\mathcal{I} \cup \mathcal{R})$. Namely, the elements of $\mathcal{I} \cap \mathcal{R}$ stay *in*, whereas the elements of $U \setminus (\mathcal{I} \cup \mathcal{R})$ stay *out*. Consequently, we will define the *inertia* for $\mathcal{I}$ and $\mathcal{R}$ as:

$$\{\mathbf{in}(a) : a \in \mathcal{I} \cap \mathcal{R}\} \cup \{\mathbf{out}(b) : b \notin (\mathcal{I} \cup \mathcal{R})\}.$$

We will assume that we need no justification for not changing the status of an element. Hence, we will use the inertia set as input to the program P. Specifically, we will eliminate the elements of the inertia from the rules in P (as they can be regarded as *true*). The modified program is then treated as a Horn program and is used to compute its least model. The resulting set of updates of the form $\mathbf{in}(a)$ and $\mathbf{out}(b)$ is then executed on $\mathcal{I}$. The order in which we execute these updates should be immaterial. Since the constraints need to be true after the revision, we will require that there is no a such that both $\mathbf{in}(a)$ and $\mathbf{out}(a)$ are computed. If the result of the process coincides with $\mathcal{R}$, we accept $\mathcal{R}$ as a P-justified revision of $\mathcal{I}$.

There are clearly similarities between P-justified revisions and stable models of logic programs. For instance, we will show that logic programs can be *identified* with revision programs consisting of rules with heads of the form $\mathbf{in}(a)$. On the other hand, revision programs can be encoded as logic programs [PT95]. So, in a sense, the converse holds as well. That is, revision programs are special logic programs. However, there are important differences. First, revision programs are explicitly designed as input-output devices. For input $\mathcal{I}$ they produce as the output a collection of P-justified revisions of $\mathcal{I}$. Secondly, and perhaps more importantly, in revision programming there is a desirable symmetry between literals $\mathbf{in}(a)$ and $\mathbf{out}(a)$ (Theorem 3.8) that is not present in logic programming, where positive and negative literals are treated differently. These and other aspects of the nature of the relationship between logic and revision programs will be discussed in detail in Section 4.

Similarly, there are differences between revision programming and DATALOG. While DATALOG programs can be viewed as input-output devices, they only add elements and

never delete. This issue was, to some extent, pursued in [AV91]. However, the semantics that we assign to revision programs is different. Secondly, in revision programming we do not distinguish between the extensional and intentional databases. In particular, the constraints may be imposed on the extensional part as well. In the case of DATALOG programs without negation this can be easily handled by means of the least cumulative fixpoint construction. With negation in the bodies allowed, the solution is no longer straightforward and other kinds of fixpoints have been used.

There are also differences between revision programming and popular formalisms that describe change by means of postulates on the effects of change, either coming from philosophy (AGM postulates, see [AGM85]) or from database theory (KM postulates, see [KM91]). In our framework, programs specify change and change is not subject to any postulates beyond those specified by the program.

In this paper we formally introduce and study revision programming. In the next section we introduce the syntax of revision programming as well as the semantics of justified revisions. We consider two equivalent definitions for P-justified revisions. One of them formalizes the ideas described above and the other one uses a modified version of Gelfond-Lifschitz approach to stable semantics of logic programs. We also introduce the necessary terminology and technical apparatus for investigating revisions.

In Section 3, we prove a number of properties of justified revisions. For instance we show that every P-justified revision of any database $\mathcal{I}$ satisfies P. We prove that P-justified revisions of $\mathcal{I}$ differ from $\mathcal{I}$ as little as possible to satisfy P. That is the symmetric difference between $\mathcal{I}$ and any P-justified revision $\mathcal{R}$ is minimal in the set of symmetric differences of $\mathcal{I}$ and models of P. We show that if $\mathcal{I}$ satisfies the constraints specified by P then $\mathcal{I}$ is its only P-justified revision. This shows that our process has the desired property that once a revision succeeds, no further change is justified. The symmetric character of **in** and **out** is highlighted in a duality result. Namely, if $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ then the complement of $\mathcal{R}$ is a P'-justified revision of the complement of $\mathcal{I}$ for a suitably constructed program P'. Moreover, the translation from P to P' is modular. We also briefly study another proposal for a semantics of revision programs given by the notion of supported revisions.

In Section 4 we show that the concept of P-justified revision generalizes that of stable model of logic program. Specifically, we show that there is a translation of logic programs to revision programs so that stable models of logic programs become precisely justified revisions of an empty database.

In Section 5 we discuss serializability of the process of P-justified revision. We show that every P-justified revision can be obtained by processing rules of P in a sequential manner. In Section 6 we discuss two classes of programs P with a property that every initial database $\mathcal{I}$ possesses a unique P-justified revision. The first of these classes, consisting of the so called *safe programs*, has also the property that *any* serialization leads to the same result, namely the unique P-justified revision. The second class, stratified programs, also produces unique revisions, but the serializations are no more arbitrary, only those that *agree* with stratification can be used. We conclude this section with a brief discussion of expressibility issues for safe revision programs and relate our

results to classic results by Smullyan [Smu68] and Apt and Blair [AB90] on expressibility of stratified logic programs.

In Section 7 we study the complexity issues of revision programming. We show that the existence problem for P-justified revisions is NP-complete. A number of other complexity results, as well as algorithms for various problems is also introduced.

The material covered in this paper has been presented in two extended abstracts: [MT94] and [MT95].

2 Preliminaries

The language of revision programming is similar to the language of logic programming. In this paper we will discuss only the propositional case. As with logic programming, the restriction to the propositional case is not essential. Our definitions and results can be lifted to the predicate case.

Let U be a denumerable set of *atoms* (a universe). Revision programming is a formalism to describe constraints on the subsets of U (databases) and provide a mechanism to enforce them. The constraints are concerned with the membership status of atoms in a database. An example of a simple constraint is: a must be present in a database. In revision programming, it is expressed by a rule

$$\mathbf{in}(a) \leftarrow .$$

Enforcing such a constraint means inserting a to the database (if a is not there already). Another example of a constraint is: a must be absent from a database. We describe it by a rule

$$\mathbf{out}(a) \leftarrow .$$

To enforce this constraint, a must be deleted from the database (if it is there).

The expressive power of revision programming goes much beyond these simple constraints. It allows the user to formulate complex constraints such as: a must not be in a database whenever b is in it and c is not. In revision programming, it is described by the rule

$$\mathbf{out}(a) \leftarrow \mathbf{in}(b), \mathbf{out}(c).$$

To enforce collections of such constraints on a database, one has to change it by inserting some atoms and removing some others. The critical question, in fact the main question studied in this paper, is: which atoms need to be inserted and which to be removed.

Formally, by a *literal* we mean an expression of the form $\mathbf{in}(a)$ or $\mathbf{out}(a)$, where a is an atom from U. The set of all literals will be denoted by *Lit*. A *revision rule* or, simply, a *rule*, is any expression of the form

$$\alpha \leftarrow \alpha_1, \ldots, \alpha_m, \tag{3}$$

where α and α_i, $1 \leq i \leq m$, are literals. The literal α is called the *head* of the rule, and the set of literals α_i, $1 \leq i \leq m$, its *body*. The head of a rule c and its body are denoted

by *head*(c) and *body*(c), respectively. If the head of a rule is of the form $\mathbf{in}(a)$, the rule is called an *in-rule*. Otherwise, it is called an *out-rule*.

A collection of rules is called a *revision program* or, simply, a *program*. The set of all literals appearing in a program P is denoted by *var*(P). The set of the heads of all rules in P is denoted by *head*(P).

The basic notion for revision programming is that of a *model* of a literal or a constraint. We say that a set of atoms $\mathcal{B} \subseteq U$ is a *model* of (or *satisfies*) a literal $\mathbf{in}(a)$, if $a \in \mathcal{B}$. Similarly, $\mathcal{B}$ is a *model* of (or *satisfies*) a literal $\mathbf{out}(b)$, if $b \notin \mathcal{B}$. A set of atoms $\mathcal{B}$ is a *model* of (or *satisfies*) a rule of the form (3) if either $\mathcal{B}$ is not a model of at least one literal α_i, or $\mathcal{B}$ is a model of α. Finally, $\mathcal{B}$ is a *model* of (or, *satisfies*) a revision program P if $\mathcal{B}$ is a model of every rule in P. The set of all models of a revision program P is denoted by $MOD(P)$. We will write $\mathcal{B} \models \alpha$, $\mathcal{B} \models c$ and $\mathcal{B} \models P$ to denote that $\mathcal{B}$ is a model of a literal α, rule c and program P, respectively.

For instance, a rule

$$\mathbf{in}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n) \tag{4}$$

is satisfied by a set of atoms $\mathcal{B}$ exactly when at least one of the following conditions holds:

1. $a \in \mathcal{B}$,
2. $a_k \notin \mathcal{B}$, for some k, $1 \leq k \leq m$,
3. $b_k \in \mathcal{B}$, for some k, $1 \leq k \leq n$.

Similarly, an out-rule

$$\mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n) \tag{5}$$

is satisfied by $\mathcal{B}$ exactly when $a \notin \mathcal{B}$, or when at least one of the conditions (2) and (3) above holds.

The main goal of this paper is to propose a semantics for revision programming. Revision programs can be viewed as operators that assign to a database a collection of its possible revisions each of which, at the very least, must be a model of P. Following this intuition, by a *semantics* for revision programming we mean any function SEM, which assigns, to every revision program P, an operator $SEM_P : \mathcal{P}(U) \rightarrow \mathcal{P}(\mathcal{P}(U))$ such that for every $\mathcal{B} \subseteq U$, $SEM_P(\mathcal{B}) \subseteq MOD(P)$.

An obvious example of a semantics is the operator SEM defined by

$$SEM_P(\mathcal{B}) = MOD(P).$$

However, it is much too weak. First, revisions of a database $\mathcal{B}$ by a program P should depend on both P and $\mathcal{B}$, and not on P only. Hence, in general, $SEM_P(\mathcal{B})$ must be a *proper* subset of the set of models of P. For instance, if a current database $\mathcal{B}$ satisfies

all the constraints in P, $SEM_P(\mathcal{B})$ should consist of $\mathcal{B}$ only (and not of all models of P) as, intuitively, there is no need for any revisions in such case.

A solution might be to define $SEM_P(\mathcal{B})$ to consist of all those models $\mathcal{B}'$ of P that differ from $\mathcal{B}$ by as little as possible, that is, for which the symmetric difference with $\mathcal{B}$ is minimal (recall that the symmetric difference is given by: $\mathcal{B}' \div \mathcal{B} = (\mathcal{B}' \setminus \mathcal{B}) \cup (\mathcal{B} \setminus \mathcal{B}')$). Formally, we define

$$MOD_{min}(P, \mathcal{B}) = \{\mathcal{B}' \in MOD(P) : \mathcal{B}' \div \mathcal{B} \text{ is minimal}\}.$$

Clearly, the operator MOD_{min} defines a semantics for revision programming. We will call it the *minimal revision* semantics. It is a counterpart of the minimal model semantics for logic programs and it suffers from a similar problem. In revision programming, by writing constraints as rules we not only express a constraint but, also, a *preferred* way to impose it. If a database $\mathcal{B}$ does not satisfy a rule (3), then all premises of the rule are satisfied but the head is not. There are two ways to guarantee that a revised version of $\mathcal{B}$ satisfies rule (3):

1. change $\mathcal{B}$ so that α is satisfied after the revision (if $\alpha = \mathbf{in}(a)$, insert a, if $\alpha = \mathbf{out}(a)$, remove a)

2. change $\mathcal{B}$ so that at least one α_k is not satisfied after the revision (if $\alpha_k = \mathbf{in}(a)$, remove a, if $\alpha_k = \mathbf{out}(a)$, insert a).

In addition to describing constraints, revision programming views rules as mechanism to infer new facts. Assuming that the premises of a rule are satisfied, the rule is used to derive its head. Consequently, it is the first way of enforcing a constraint that is preferred.

Example 2.1 Consider a database $\mathcal{B} = \{a, b\}$ and a program $P = \{\mathbf{out}(b) \leftarrow \mathbf{in}(a)\}$. Clearly, $\mathcal{B}$ is not a model of P. In order to satisfy P, we have two possibilities: (1) remove b from $\mathcal{B}$, and (2) remove a from $\mathcal{B}$. Each of these possibilities leads to a model of P minimally differing from $\mathcal{B}$. However, the first one is preferred, as it reflects the intuition of a rule as an inference mechanism. Hence, we should require that $SEM_P(\mathcal{B}) = \{\{a\}\}$.

The analogies with logic programming are quite obvious. We will study the correspondence in more detail in Section 4.

We will now introduce our proposal for the semantics of revision programs. We will start with more terminology. Let $\mathcal{B} \subseteq U$ be a set of atoms. We define

$$\mathcal{B}^c = \{\mathbf{in}(a) \colon a \in \mathcal{B}\} \cup \{\mathbf{out}(a) \colon a \notin \mathcal{B}\}.$$

For any set of literals L, we define

$$L^+ = \{a \in U \colon \mathbf{in}(a) \in L\}$$

and

$$L^- = \{a \in U : \mathbf{out}(a) \in L\}.$$

We call a set L of literals *coherent* if $L^- \cap L^+ = \emptyset$. Clearly, a coherent set of literals determines a revision of a database $\mathcal{B}$ as it specifies necessary insertions and deletions to be performed. Namely, the result of the revisions determined by L is the database $(\mathcal{B} \setminus L^-) \cup L^+$. We will denote it by $\mathcal{B} \oplus L$. That is,

$$\mathcal{B} \oplus L = (\mathcal{B} \setminus L^-) \cup L^+.$$

Notice that if a set L of literals is coherent, then $(\mathcal{B} \setminus L^-) \cup L^+ = (\mathcal{B} \cup L^+) \setminus L^-$. More generally, the order in which insertions and deletions specified by L are executed is immaterial. However, if L is incoherent, then $(\mathcal{B} \setminus L^-) \cup L^+ \neq (\mathcal{B} \cup L^+) \setminus L^-$. Thus, the order in which the changes are made becomes crucial. Consequently, in such case, the revision by a set of literals becomes ill-defined. This is the reason why we do not consider revisions specified by incoherent sets of literals.

The following lemma summarizes basic properties of the notion of model and the operator $\oplus$.

Lemma 2.2 *Let L be a set of literals and let $\mathcal{B}$ be a database.*

1. *If $\mathcal{B} \models L$, then L is coherent and $\mathcal{B} \oplus L = \mathcal{B}$.*
2. *Let L be a coherent set of literals. If $L \subseteq L'$ and $\mathcal{B} \oplus L \models L'$, then $\mathcal{B} \oplus L = \mathcal{B} \oplus L'$.*
3. *Let L be coherent. If $\alpha \in L$, then $\mathcal{B} \oplus L \models \alpha$. If $\mathcal{B} \oplus L \models \alpha$ and $\mathcal{B} \not\models \alpha$, then $\alpha \in L$.* □

Our proposal for a semantics for revision programming is based on two key concepts: *necessary change* and *inertia set*. We will now introduce these notions and study their properties.

Example 2.3 Consider the program $P = \{\mathbf{in}(c) \leftarrow , \mathbf{out}(b) \leftarrow \mathbf{in}(c)\}$ and the initial database $\mathcal{B} = \{a, b\}$. Since c must be inserted unconditionally, b — whose removal is conditioned only upon believing in c — must be removed. Literals $\mathbf{in}(c)$ and $\mathbf{out}(b)$ form the *necessary change* determined by P.

In this example, the intuition behind the term "necessary change" is that no matter what the initial database is, the actions described by it (insert some objects, eliminate some objects) will have to be performed. This intuition is formalized as follows.

Definition 2.4 (Necessary change) *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 $\mathbf{in}(a)$ and $\mathbf{out}(b)$.*

One can develop the notion of necessary change in the language of operator theory. Namely, one can assign to a revision program P the corresponding *one-step revision operator* and then prove that the necessary change is its least fixpoint. This one-step revision operator coincides, in fact, with van Emden-Kowalski [vEK76] operator for P treated as a Horn program (Definition 2.4). Hence, it is monotone and compact and, consequently, the fixpoint exists and is reached in at most ω steps.

Since the notion of necessary change is defined as a least model of a certain Horn program, we will recall some well-known properties of propositional Horn programs. We will not prove them, although, to the best of our knowledge, they were not reported in the literature.

Let P be a propositional Horn program. By $LM(P)$ we denote a least model of P. Let us define

$$P^u = \{c \in P : body(c) \subseteq LM(P)\}.$$

Intuitively, P^u consists of all those rules in P which "fire" (are *used*) during the construction of the least model of P. In particular,

$$head(P^u) = LM(P).$$

Lemma 2.5 *Let P be a Horn program and let $P = P_1 \cup P_2$. If*

1. *$P_1 \cap P_2 = \emptyset$, and*
2. *for every rule $c \in P_2$, $body(c) \not\subseteq head(P_1)$,*

then $P^u = P_1^u$ and $LM(P) = LM(P_1)$. If, in addition, $P_1^u = P_1$ then $P^u = P_1$ and $LM(P) = LM(P_1) = head(P_1)$. □

Let P be a Horn program and let $\mathcal{B}$ be a set of atoms. By $P|\mathcal{B}$ we denote the Horn program obtained from P by eliminating from the body of each rule all atoms that are in $\mathcal{B}$.

Lemma 2.6 *Let P be a propositional Horn program over a denumerable set of atoms U. Let P' be a subset of P and $\mathcal{B}$ be a set of atoms such that*

1. *for every rule $c \in P \setminus P'$, $body(c) \not\subseteq head(P') \cup \mathcal{B}$, and*
2. *there is an enumeration $\{c_t : 1 \leq t < n\}$, where n is a non-negative integer or $n = \omega$, of the rules in P' such that for every t, $1 \leq t < n$,*

$$body(c_t) \subseteq head(\{c_q : 1 \leq q < t\}) \cup \mathcal{B}.$$

Then, $LM(P|\mathcal{B}) = head(\{c_t : 1 \leq t < n\})$. □

Lemma 2.7 *Let P be a propositional Horn program over a denumerable set of atoms U. Let $\mathcal{B}$ be a set of atoms and let P' be a subset of P such that $(P|\mathcal{B})^u = P'|\mathcal{B}$. There exists an enumeration $\{c_k : 1 \leq k < n\}$ of P', where n is a non-negative integer or $n = \omega$, such that*

1. *for every rule* $c \in P \setminus P'$, $body(c) \not\subseteq head(P') \cup \mathcal{B}$, *and*

2. *for every* k, $1 \leq k < n$, $body(c_k) \subseteq head(\{c_q : 1 \leq q < k\}) \cup \mathcal{B}$. □

In the paper, we will use the notation introduced above as well as Lemmas 2.5, 2.6 and 2.7 for revision programs treated as propositional Horn programs.

The second key concept is that of the inertia set. Let P be a revision program. Consider an initial database $\mathcal{I}$. Assume that $\mathcal{R}$ is a revision of $\mathcal{I}$ by P. Clearly, P must provide justification for the insertions of the elements in $\mathcal{R} \setminus \mathcal{I}$ and the deletions of the elements in $\mathcal{I} \setminus \mathcal{R}$. Since the status of all the other elements remains the same, no other justifications are needed. In fact, we will use the elements whose status does not change (formally described by the inertia set), in combination with the program P, to provide justification for all the changes necessary to transform $\mathcal{I}$ into $\mathcal{R}$.

Let us define the *inertia set* for the pair $(\mathcal{I}, \mathcal{R})$ as follows:

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

The following lemma gathers several simple properties of the inertia set.

Lemma 2.8 *Let* $\mathcal{I}$, $\mathcal{I}'$ *and* $\mathcal{R}$ *be databases, let* L *be a set of literals and let* α *be a literal.*

1. $\alpha \in I(\mathcal{I}, \mathcal{R})$ *if and only if* $\mathcal{I} \models \alpha$ *and* $\mathcal{R} \models \alpha$,

2. $I(\mathcal{I}, \mathcal{R}) \subseteq I(\mathcal{I}', \mathcal{R})$ *if and only if* $\mathcal{R} \div \mathcal{I}' \subseteq \mathcal{R} \div \mathcal{I}$,

3. *If* $I(\mathcal{I}, \mathcal{R}) \subseteq I(\mathcal{I}', \mathcal{R})$, L *is coherent and* $\mathcal{R} = \mathcal{I} \oplus L$, *then* $\mathcal{R} = \mathcal{I}' \oplus L$. □

Part (1) of the lemma expresses a basic intuition behind the inertia set. It consists of those literals that are satisfied by both $\mathcal{I}$ and $\mathcal{R}$. Part (2) shows that the larger the inertia set the "closer" the two databases are (and conversely). Finally, part (3) shows that if $\mathcal{R}$ is obtained by revising $\mathcal{I}$ by L and if $\mathcal{I}'$ is "closer" to $\mathcal{R}$ than $\mathcal{I}$, then revising $\mathcal{I}'$ by L also yields $\mathcal{R}$.

We will use the literals in $I(\mathcal{I}, \mathcal{R})$ to simplify P (they need not to be justified by P, as they are satisfied by both $\mathcal{I}$ and $\mathcal{R}$ and can be assumed to hold). By the *reduct of* P *with respect to* $(\mathcal{I}, \mathcal{R})$, denoted by $P_{\mathcal{I},\mathcal{R}}$, we mean the revision program obtained from P by eliminating from the body of each rule in P all literals in $I(\mathcal{I}, \mathcal{R})$. That is, in the notation introduced earlier for the Horn programs,

$$P_{\mathcal{I},\mathcal{R}} = P | I(\mathcal{I}, \mathcal{R}).$$

The necessary change of the program $P_{\mathcal{I},\mathcal{R}}$ provides a justification for some insertions and deletions. These are exactly the changes that are justified by P in the context of the pair of databases $(\mathcal{I}, \mathcal{R})$.

Definition 2.9 (Justified Revision) Let P be a revision program and let $\mathcal{I}$ and $\mathcal{R}$ be databases. If $NC(P_{\mathcal{I},\mathcal{R}})$ is coherent and

$$\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$$

then $\mathcal{R}$ is called a *P-justified revision* of $\mathcal{I}$.

For every revision program P and every database $\mathcal{B}$, by $JR_P(\mathcal{B})$ we denote the set of all P-justified revisions of $\mathcal{B}$. We propose the operator JR as a semantics for revision programs. At this point it is not at all clear that $JR_P(\mathcal{B}) \subseteq MOD_P(\mathcal{B})$ (that is, that JR indeed defines a semantics for revision programming). We will later show that it is the case (Theorem 3.1).

We will now illustrate the notions of necessary change, inertia set, reduct and P-justified revision.

Example 2.10 Consider the program $P = \{\mathbf{in}(a) \leftarrow \mathbf{out}(b), \mathbf{in}(b) \leftarrow \mathbf{out}(a)\}$. Assume that $\mathcal{I} = \emptyset$ and $\mathcal{R} = \{a, b\}$. Clearly,

$$I(\mathcal{I}, \mathcal{R}) = \emptyset.$$

Consequently, $P_{\mathcal{I},\mathcal{R}} = P$ and

$$NC(P_{\mathcal{I},\mathcal{R}}) = \emptyset.$$

Hence, P does not justify any changes in the context of $(\mathcal{I}, \mathcal{R})$. Therefore, $\mathcal{R} \neq \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$ and, consequently, $\mathcal{R}$ is not a P-justified revision of $\mathcal{I}$.

Assume now that $\mathcal{I}$ is as before and that $\mathcal{R} = \{a\}$. Now,

$$I(\mathcal{I}, \mathcal{R}) = \{\mathbf{out}(b)\} \quad \text{and} \quad P_{\mathcal{I},\mathcal{R}} = \{\mathbf{in}(a) \leftarrow,\ \mathbf{in}(b) \leftarrow \mathbf{out}(a)\}.$$

Clearly,

$$NC(P_{\mathcal{I},\mathcal{R}}) = \{\mathbf{in}(a)\}.$$

Since $NC(P_{\mathcal{I},\mathcal{R}})$ is coherent and $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$, $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$.

The same reasoning shows that $\mathcal{R} = \{b\}$ is a P-justified revision of $\mathcal{I}$ and that $\mathcal{R} = \emptyset$ is not a P-justified revision of $\mathcal{I}$.

Example 2.11 Let $U = \{a, b\}$. Let $P = \{\mathbf{out}(a) \leftarrow \mathbf{in}(a)\}$. Consider a database $\mathcal{I} = \{a\}$. Then, no set of atoms is a P-justified revision of $\{a\}$. For example, let $\mathcal{R} = \emptyset$. Then, $I(\mathcal{I}, \mathcal{R}) = \emptyset$, $P_{\mathcal{I},\mathcal{R}} = P$ and $NC(P_{\mathcal{I},\mathcal{R}}) = \emptyset$. Clearly, $NC(P_{\mathcal{I},\mathcal{R}})$ is coherent but $\mathcal{R} \neq \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$. Similarly, we show that none of the remaining subsets of U ($\{a\}$, $\{b\}$ and $\{a, b\}$) is a P-justified revision of $\mathcal{I}$.

In the same time, if $\mathcal{I} = \emptyset$, then $\mathcal{R} = \emptyset$ is the only P justified revision of $\mathcal{I}$.

These two examples show that given a revision program P, a database $\mathcal{B}$ can have no, exactly one, or many justified revisions. Especially important, from the point of view of practical database applications are those revision programs that, for every input

database, uniquely determine its revision. We exhibit two classes of such programs later in this paper.

We will now provide an alternative definition of P-justified revisions. It is based on a different notion of reduct — a counterpart of Gelfond-Lifschitz reduct in logic programming [GL88].

Let P be a revision program and let $\mathcal{I}$ and $\mathcal{R}$ be two databases. The *GL-reduct of P with respect to $(\mathcal{I}, \mathcal{R})$* is defined in two stages:

Stage 1: Eliminate from P every rule whose body is not satisfied by $\mathcal{R}$. Denote the resulting program by $P_{\mathcal{R}}$.

Stage 2: From the body of each rule in $P_{\mathcal{R}}$ eliminate each literal that is satisfied by $\mathcal{I}$. Denote the resulting program by $P_{\mathcal{R}}|\mathcal{I}$ (this is the GL-reduct of P with respect to $(\mathcal{I}, \mathcal{R})$).

(Observe that $P_{\mathcal{R}}|\mathcal{I} = P_{\mathcal{R}}|I(\mathcal{I}, \mathcal{R})$.)

A comment is warranted here. In the original paper by Gelfond and Lifschitz, the first stage of the reduction is different from the one described here. Namely, Gelfond and Lifschitz eliminate from P only those rules that have at least one *negative* literal in the body not satisfied by a hypothetical stable model (a counterpart of $\mathcal{R}$). In our approach, we eliminate all those rules that have at least one literal, *positive* or *negative*, not satisfied by $\mathcal{R}$. This is an important point. As we will see there is a high degree of symmetry in revision programming — positive and negative literals are treated in the same way. The original definition of the Gelfond-Lifschitz reduct, which treats positive and negative atoms differently, was not suitable as a template for a reduct of revision programs. However, the first step in the construction of Gelfond and Lifschitz can be modified. One can eliminate all those rules whose *body* is not satisfied by the hypothetical stable model. The notion of the reduct changes but the notion of the stable model remains the same! It is this approach that is generalized here to the case of revision programs.

The following theorem ties together the notions of reduct and GL-reduct for revision programs, and shows that each can be used to define the notion of P-justified revision.

Theorem 2.12 *Let P be a revision program and let $\mathcal{I}$ and $\mathcal{R}$ be two databases. The following two conditions are equivalent:*

(R1) *$NC(P_{\mathcal{I},\mathcal{R}})$ is coherent and $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$,*

(R2) *$NC(P_{\mathcal{R}}|\mathcal{I})$ is coherent and $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{R}}|\mathcal{I})$.*

Proof: Assume (R1). We will first show that

$$P^u_{\mathcal{I},\mathcal{R}} = P_{\mathcal{R}}|\mathcal{I}. \tag{6}$$

Consider a rule c

$$\alpha \leftarrow \alpha_1, \ldots, \alpha_n$$

from $P^u_{\mathcal{I},\mathcal{R}}$. By the definition of $P^u_{\mathcal{I},\mathcal{R}}$,

$$\alpha_1, \ldots, \alpha_n \in NC(P_{\mathcal{I},\mathcal{R}}).$$

Since $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$, Lemma 2.2 implies that

$$\mathcal{R} \models \alpha_1, \ldots, \alpha_n. \tag{7}$$

By the definition of $P_{\mathcal{I},\mathcal{R}}$, none of the literals α_j is in $I(\mathcal{I},\mathcal{R})$. Hence, by Lemma 2.8 and (7), for every j, $1 \leq j \leq n$,

$$\mathcal{I} \not\models \alpha_j. \tag{8}$$

Since $c \in P_{\mathcal{I},\mathcal{R}}$, there are literals $\beta_1, \ldots, \beta_k \in I(\mathcal{I},\mathcal{R})$ such that the rule c', of the form

$$\alpha \leftarrow \alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_k,$$

is in P. By Lemma 2.8, $\mathcal{R} \models \beta_1, \ldots, \beta_k$. Hence, by (7), $c' \in P_{\mathcal{R}}$. Moreover, also by Lemma 2.8, we have that $\mathcal{I} \models \beta_1, \ldots, \beta_k$. Hence, by (8), $\alpha \leftarrow \alpha_1, \ldots, \alpha_n$ is in $P_{\mathcal{R}}|\mathcal{I}$. This proves that $P^u_{\mathcal{I},\mathcal{R}} \subseteq P_{\mathcal{R}}|\mathcal{I}$.

To prove the converse inclusion, consider a rule c

$$\alpha \leftarrow \alpha_1, \ldots, \alpha_n$$

from $P_{\mathcal{R}}|\mathcal{I}$.

By the definition of $P_{\mathcal{R}}|\mathcal{I}$, for every j, $1 \leq j \leq n$, $\mathcal{R} \models \alpha_j$ and $\mathcal{I} \not\models \alpha_j$. Hence, none of α_j is in $I(\mathcal{I},\mathcal{R})$. Moreover, there are literals $\beta_1, \ldots, \beta_k$ such that $\mathcal{I} \models \beta_j$, $1 \leq j \leq k$, and the rule c', of the form

$$\alpha \leftarrow \alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_k,$$

is in $P_{\mathcal{R}}$. It follows that $\mathcal{R} \models \beta_j$, $1 \leq j \leq k$.

Now, it is easy to see that β_j, $1 \leq j \leq k$, are the only literals in the body of c' that belong to $I(\mathcal{I},\mathcal{R})$. Consequently, $c \in P_{\mathcal{I},\mathcal{R}}$. Recall that $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$, and that for every j, $1 \leq j \leq n$, $\mathcal{R} \models \alpha_j$ and $\mathcal{I} \not\models \alpha_j$. By Lemma 2.2, for every j, $1 \leq j \leq n$, $\alpha_j \in NC(P_{\mathcal{I},\mathcal{R}})$. Consequently, $c \in P^u_{\mathcal{I},\mathcal{R}}$.

Thus, we have proved (6). It follows that

$$NC(P_{\mathcal{I},\mathcal{R}}) = NC(P^u_{\mathcal{I},\mathcal{R}}) = NC(P_{\mathcal{R}}|\mathcal{I}).$$

Consequently, $NC(P_{\mathcal{R}}|\mathcal{I})$ is coherent and $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{R}}|\mathcal{I})$.

Assume now that (R2) holds. We will prove (R1). We will show that also in this case the identity (6) holds. First, recall that $P_{\mathcal{R}}|\mathcal{I}$ is obtained from $P_{\mathcal{R}}$ by eliminating from the body of each rule all literals satisfied by $\mathcal{I}$. Since all literals in the body of each rule of $P_{\mathcal{R}}$ are satisfied by $\mathcal{R}$, the result is the same when we eliminate from the body of each rule in $P_{\mathcal{R}}$ the literals satisfied both by $\mathcal{I}$ and $\mathcal{R}$, that is, the literals in $I(\mathcal{I},\mathcal{R})$. It follows that

$$P_{\mathcal{R}}|\mathcal{I} = (P_{\mathcal{R}})_{\mathcal{I},\mathcal{R}} \subseteq P_{\mathcal{I},\mathcal{R}}.$$

Hence, $NC(P_{\mathcal{R}}|\mathcal{I}) \subseteq NC(P_{\mathcal{I},\mathcal{R}})$.

Consider a rule c

$$\alpha \leftarrow \alpha_1, \ldots, \alpha_n$$

from $P_{\mathcal{R}}|\mathcal{I}$. Then, for every j, $1 \leq j \leq n$, $\mathcal{R} \models \alpha_j$ and $\mathcal{I} \not\models \alpha_j$. Since $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{R}}|\mathcal{I})$, $\alpha_j \in NC(P_{\mathcal{R}}|\mathcal{I})$ (Lemma 2.2). Consequently, $(P_{\mathcal{R}}|\mathcal{I})^u = P_{\mathcal{R}}|\mathcal{I}$.

Observe now that every rule from $P_{\mathcal{I},\mathcal{R}} \setminus (P_{\mathcal{R}}|\mathcal{I})$ has at least one literal in the body that is not satisfied by $\mathcal{R}$. Since, $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{R}}|\mathcal{I})$, by Lemma 2.2 it follows that every rule from $P_{\mathcal{I},\mathcal{R}} \setminus (P_{\mathcal{R}}|\mathcal{I})$ has at least one literal in the body that does not belong to $NC(P_{\mathcal{R}}|\mathcal{I})$. By Lemma 2.5, it follows that $P^u_{\mathcal{I},\mathcal{R}} = P_{\mathcal{R}}|\mathcal{I}$. That is, (6) holds.

The rest of the proof is almost as before. We have $NC(P_{\mathcal{I},\mathcal{R}}) = NC(P^u_{\mathcal{I},\mathcal{R}}) = NC(P_{\mathcal{R}}|\mathcal{I})$. Hence, $NC(P_{\mathcal{I},\mathcal{R}})$ is coherent and, therefore, $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$. □

The analysis of the proof of Theorem 2.12 implies another important result. It will be used frequently throughout the paper.

Theorem 2.13 *Let P be a revision program and let $\mathcal{R}$ be a P-justified revision of $\mathcal{I}$. Then, $P^u_{\mathcal{I},\mathcal{R}} = P_{\mathcal{R}}|\mathcal{I}$ and $NC(P_{\mathcal{I},\mathcal{R}}) = NC(P_{\mathcal{R}}|\mathcal{I}) = head(P_{\mathcal{R}})$.* □

Finally, we will state yet another characterization of justified revisions.

Theorem 2.14 *The following conditions are equivalent:*

1. *A database $\mathcal{R}$ is a P-justified revision of a database $\mathcal{I}$,*
2. *$NC(P \cup \{\alpha \leftarrow : \alpha \in I(\mathcal{I},\mathcal{R})\}) = \mathcal{R}^c$,*
3. *$NC(P_{\mathcal{I},\mathcal{R}}) \cup I(\mathcal{I},\mathcal{R}) = \mathcal{R}^c$.*

Proof: It is easy to see that

$$NC(P \cup \{\alpha \leftarrow : \alpha \in I(\mathcal{I},\mathcal{R})\}) = NC(P_{\mathcal{I},\mathcal{R}}) \cup I(\mathcal{I},\mathcal{R}).$$

Hence, to prove the theorem, one only has to show the equivalence of (1) and (3). Assume (3). Since $\mathcal{R}^c$ is coherent, $NC(P_{\mathcal{I},\mathcal{R}})$ is coherent, too. We will now show that $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$. Let $a \in \mathcal{R}$. Then $\mathbf{in}(a) \in \mathcal{R}^c$ and, consequently, $\mathbf{in}(a) \in NC(P_{\mathcal{I},\mathcal{R}}) \cup I(\mathcal{I},\mathcal{R})$. If $\mathbf{in}(a) \in NC(P_{\mathcal{I},\mathcal{R}})$, then $a \in \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$. So, assume that $\mathbf{in}(a) \in I(\mathcal{I},\mathcal{R})$. It follows that $a \in \mathcal{I}$. Since $\mathbf{out}(a) \notin NC(P_{\mathcal{I},\mathcal{R}})$ (recall that $\mathbf{out}(a) \notin \mathcal{R}^c$), it follows that $a \in \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$. Hence, $\mathcal{R} \subseteq \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$. The converse inclusion can be proved similarly. Hence, (1) follows. The proof that (1) implies (3) is similar and is left to the reader. □

3 Basic results

In this section we present a number of fundamental properties of revision programming. All these results are very natural and indicate that the notion of P-justified revision corresponds to the intuitions normally associated with the process of change.

Our first result shows that the notion of a P-justified revision indeed specifies a semantics for revision programming, that is, that P-justified revisions are models of a program P. In the terminology of Section 2, we show that $JR_P(\mathcal{B}) \subseteq MOD(P)$.

Theorem 3.1 *Let P be a revision program and let $\mathcal{I}$ be a database. If a database $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$, then $\mathcal{R}$ is a model of P.*

Proof: Since $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$, $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{R}}|\mathcal{I})$. By Lemma 2.2 and Theorem 2.13, it follows that $\mathcal{R} \models \mathit{head}(P_{\mathcal{R}})$. Consequently, $\mathcal{R} \models P_{\mathcal{R}}$. Since for every rule $c \in P \setminus P_{\mathcal{R}}$, $\mathcal{R} \not\models \mathit{body}(c)$, $\mathcal{R} \models c$. Hence, $\mathcal{R} \models P$. □

A common feature of knowledge representation formalisms is the *confirmation of evidence* property. If a belief set is selected on the basis of some data and if additional data, consistent with this belief set is received, then there is no need to change the belief set (although new belief sets may become possible at this point). This new evidence can come as new facts already present in the belief set, and as new rules that are satisfied by the belief set. The first of these possibilities was studied in the case of default logic and logic programming [Rei80, MT93a]. The second one has not been explicitly studied in the literature so far. We will now prove two versions of confirmation of evidence property for revision programming.

In the next result, the assumption $\mathcal{R} \div \mathcal{B} \subseteq \mathcal{R} \div \mathcal{I}$ means that $\mathcal{B}$ is "closer" to $\mathcal{R}$ than $\mathcal{I}$. That is, it contains additional confirmation for the choice of $\mathcal{R}$ as the revision of $\mathcal{I}$.

Theorem 3.2 *Let $\mathcal{R}$ be a P-justified revision of $\mathcal{I}$ and let $\mathcal{B}$ be a database such that $\mathcal{R} \div \mathcal{B} \subseteq \mathcal{R} \div \mathcal{I}$. Then, $\mathcal{R}$ is a P-justified revision of $\mathcal{B}$.*

Proof: Consider a rule $c \in P_{\mathcal{R}}$. Let α be a literal in the body of c. Assume that $\mathcal{B} \not\models \alpha$. Since $\mathcal{R} \models \alpha$ (recall that $c \in P_{\mathcal{R}}$), it follows that $\alpha \notin I(\mathcal{B}, \mathcal{R})$. By Lemma 2.8, $\alpha \notin I(\mathcal{I}, \mathcal{R})$. Since $\mathcal{R} \models \alpha$, it follows that $\mathcal{I} \not\models \alpha$. Consequently, for every rule $c \in P_{\mathcal{R}}|\mathcal{B}$, its body is a subset of the body of the corresponding rule in $P_{\mathcal{R}}|\mathcal{I}$. Hence,

$$NC(P_{\mathcal{R}}|\mathcal{I}) \subseteq NC(P_{\mathcal{R}}|\mathcal{B}) \subseteq \mathit{head}(P_{\mathcal{R}}).$$

By Theorem 2.13, $NC(P_{\mathcal{R}}|\mathcal{I}) = NC(P_{\mathcal{R}}|\mathcal{B})$. Hence, $NC(P_{\mathcal{R}}|\mathcal{B})$ is coherent. By Lemma 2.8(3), $\mathcal{R} = \mathcal{B} \oplus NC(P_{\mathcal{R}}|\mathcal{B})$. □

The next result deals with the situation when additional evidence comes in the form of new revision rules.

Theorem 3.3 *Let $\mathcal{R}$ be a P-justified revision of $\mathcal{I}$. Assume that P' is a revision program such that $\mathcal{R} \models P'$. Then, $\mathcal{R}$ is a $(P \cup P')$-justified revision of $\mathcal{I}$.*

Proof: Define $P'' = P \cup P'$. Then

$$P''_{\mathcal{R}}|\mathcal{I} = (P_{\mathcal{R}}|\mathcal{I}) \cup (P'_{\mathcal{R}}|\mathcal{I}).$$

By Theorem 2.13, $NC(P_{\mathcal{R}}|\mathcal{I}) = head(P_{\mathcal{R}})$. Hence,

$$head(P_{\mathcal{R}}) \subseteq NC(P''_{\mathcal{R}}|\mathcal{I}) \subseteq head(P_{\mathcal{R}}) \cup head(P'_{\mathcal{R}}).$$

Since $\mathcal{R}$ is a model of P, $\mathcal{R} \models head(P_{\mathcal{R}})$. Since $\mathcal{R} \models P'$, $\mathcal{R} \models head(P'_{\mathcal{R}})$. Consequently, $\mathcal{R} \models NC(P''_{\mathcal{R}}|\mathcal{I})$. Thus, $NC(P''_{\mathcal{R}}|\mathcal{I})$ is coherent and

$$\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{R}}|\mathcal{I}) = \mathcal{I} \oplus NC(P''_{\mathcal{R}}|\mathcal{I})$$

(by Lemma 2.2). □

Theorem 3.3 implies the following corollary.

Corollary 3.4 *Let P be a revision program. A database $\mathcal{R}$ is a $P_{\mathcal{R}}$-justified revision of $\mathcal{I}$ if and only if $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$.* □

Another intuitive principle of revision is that if the current database satisfies all desired constraints then no change is necessary. The next result shows that under the semantics of justified revisions not only no change is necessary but, if there are no other constraints, no change can be justified.

Theorem 3.5 *If a database $\mathcal{B}$ satisfies a revision program P then $\mathcal{B}$ is a unique P-justified revision of $\mathcal{B}$.*

Proof: Observe first that $NC(P_{\mathcal{B}}|\mathcal{B}) \subseteq head(P_{\mathcal{B}})$. Since $\mathcal{B}$ is a model of P, we have $\mathcal{B} \models head(P_{\mathcal{B}})$. Consequently, $NC(P_{\mathcal{B}}|\mathcal{B})$ is coherent and $\mathcal{B} = \mathcal{B} \oplus NC(P_{\mathcal{B}}|\mathcal{B})$ (Lemma 2.2). Hence, $\mathcal{B}$ is its own P-justified revision.

Consider now a P-justified revision $\mathcal{B}'$ of $\mathcal{B}$. Consider a rule $c \in P_{\mathcal{B}'}$ given by

$$\alpha \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n).$$

There are two possibilities.

Case 1. $\mathcal{B}$ satisfies the body of c. Since $\mathcal{B}$ is a model of P, $\mathcal{B}$ satisfies α. In addition, $\alpha \leftarrow$ belongs to $P_{\mathcal{B}'}|\mathcal{B}$.

Case 2. $\mathcal{B}$ does not satisfy the body of c. Then, the rule c' that c contributes to $P_{\mathcal{B}'}|\mathcal{B}$ (that is, the rule obtained from c by eliminating from its body all literals satisfied by $\mathcal{B}$) has a nonempty body. In fact, none of the elements in the body of c' is satisfied by $\mathcal{B}$.

Hence, $P_{\mathcal{B}'}|\mathcal{B}$ consists of rules of two types: (1) rules with the empty body and with the head satisfied by $\mathcal{B}$, and (2) rules with a nonempty body in which no element is satisfied by $\mathcal{B}$. It follows that $\mathcal{B} \models NC(P_{\mathcal{B}'}|\mathcal{B})$. Hence, by Lemma 2.2, $\mathcal{B} \oplus NC(P_{\mathcal{B}'}|\mathcal{B}) = \mathcal{B}$. Since $\mathcal{B}'$ is a P-justified revision of $\mathcal{B}$, $\mathcal{B}' = \mathcal{B} \oplus NC(P_{\mathcal{B}'}|\mathcal{B})$. Hence, $\mathcal{B} = \mathcal{B}'$ and $\mathcal{B}$ is a unique P-justified revision of $\mathcal{B}$. □

Major nonmonotonic reasoning systems and several theories of belief revision and database update satisfy some version of the minimality (parsimony) principle. For example, stable models of a logic program P are minimal models of P and extensions of a default theory (D, W) are minimal theories closed under (D, W) (see [MT93b]). In a modal nonmonotonic logic $\mathcal{S}$, $\mathcal{S}$-expansions can be characterized in terms of Kripke models satisfying some minimality criteria ([Sch92, MT93b]). Similarly, in the case of theories of belief revision and database update, we require that theories (databases) after revision or update differ from the initial ones by "as little as possible".

The process of change described by P-justified revisions has a strong proof-theoretic flavor (we have an *a posteriori* valid justification of any change in status of every element). Consequently, the notion of a justified revision also satisfies certain natural minimality criterion. Given two sets $\mathcal{R}$ and $\mathcal{I}$, one can describe how much they differ by means of their symmetric difference $\mathcal{R} \div \mathcal{I}$ or, equivalently, by means of the corresponding inertia set (Lemma 2.8). Intuitively, P-justified revisions of a database should differ from the database by as little as possible. Our next result formally describes a minimality condition satisfied by justified revisions.

Theorem 3.6 *Let P be a revision program and let $\mathcal{I}$ be a database. If $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$, then $\mathcal{R} \div \mathcal{I}$ is minimal in the family $\{\mathcal{B} \div \mathcal{I} : \mathcal{B}$ is a model of $P\}$.*

Proof: Assume that $\mathcal{B}$ is a model of P and that $\mathcal{B} \div \mathcal{I} \subseteq \mathcal{R} \div \mathcal{I}$. It follows that $\mathcal{R} \div \mathcal{B} \subseteq \mathcal{R} \div \mathcal{I}$. By Theorem 3.2, $\mathcal{R}$ is a P-justified revision of $\mathcal{B}$. Since $\mathcal{B}$ is a model of P, by Theorem 3.5, $\mathcal{R} = \mathcal{B}$. □

Theorem 3.6 has a corollary which generalizes a well-known property that all stable models (extensions) of a logic program (default theory) form an antichain.

Corollary 3.7 *Let P be a revision program and let $\mathcal{I}$ be a database. If $\mathcal{R}$ and $\mathcal{R}'$ are P-justified revisions of $\mathcal{I}$ and $\mathcal{R} \div \mathcal{I} \subseteq \mathcal{R}' \div \mathcal{I}$, then $\mathcal{R} = \mathcal{R}'$.* □

We will now study the notion of a dual revision program. Each database $\mathcal{B}$ uniquely determines its complement $\overline{\mathcal{B}} = U \setminus \mathcal{B}$. We will now show that the chain of transitions

$$\overline{\mathcal{I}} \mapsto \mathcal{I} \stackrel{P}{\mapsto} \mathcal{R} \mapsto \overline{\mathcal{R}}$$

can be performed directly by a single transformation

$$\overline{\mathcal{I}} \stackrel{P'}{\mapsto} \overline{\mathcal{R}}$$

for a suitably constructed program P'.

For a literal $\mathbf{in}(a)$, its *dual* is the literal $\mathbf{out}(a)$. Similarly, the *dual* of $\mathbf{out}(a)$ is $\mathbf{in}(a)$. The dual of a literal α is denoted by α^D. For a set of literals L, we define $L^D = \{\alpha^D : \alpha \in L\}$. Given a revision program P, let us define the *dual* of P (P^D in symbols) to be the revision program obtained form P by simultaneously replacing all occurrences of all literals by their duals. It is easy to see that whatever has to be added

to $\mathcal{I}$ according to revisions specified by P has to be removed from $\overline{\mathcal{I}}$ according to P^D. Similarly, whatever has to be removed from $\mathcal{I}$ according to P has to be added to $\overline{\mathcal{I}}$ according to P^D. Hence, in revision programming there is duality between **in** and **out** operators.

Theorem 3.8 (Duality Theorem) *Let P be a revision program and let $\mathcal{I}$ be a database. Then, $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $\overline{\mathcal{R}}$ is a P^D-justified revision of $\overline{\mathcal{I}}$.*

Proof: Observe that for every database $\mathcal{B}$

$$\overline{\mathcal{B}}^c = (\mathcal{B}^c)^D.$$

Observe also that for every two databases $\mathcal{I}$ and $\mathcal{R}$,

$$I(\overline{\mathcal{I}}, \overline{\mathcal{R}}) = I(\mathcal{I}, \mathcal{R})^D.$$

Finally, notice that for every revision program P,

$$NC(P^D) = (NC(P))^D.$$

All these observations and Theorem 2.14 imply the assertion. □

Finally, we will discuss some properties of necessary change and the notion of coherence. The next result shows that updates implied by the necessary change of a program P are consistent with the models of P.

Theorem 3.9 *Let P be a revision program. For every model $\mathcal{M}$ of P, $NC^+(P) \subseteq \mathcal{M}$ and $NC^-(P) \cap \mathcal{M} = \emptyset$.*

Proof: A set $\mathcal{M}$ of atoms is a model of a revision program if and only if $\mathcal{M}^c$ is a model of a (Horn) logic program obtained from P by regarding each revision literal as a distinct propositional atom. Let $\mathcal{M}$ be a model of P. By the definition of $NC(P)$, $NC(P) \subseteq \mathcal{M}^c$. Hence, the assertion follows. □

Corollary 3.10 *If a revision program P has a model then $NC(P)$ is coherent.* □

The converse to Corollary 3.10 fails. For example, consider a program $P = \{\mathbf{in}(a) \leftarrow \mathbf{out}(a),\ \mathbf{out}(a) \leftarrow \mathbf{in}(a)\}$. Clearly, P has no models. In the same time, $NC(P) = \emptyset$. Hence, it is coherent. However, the notion of coherence can be given a complete characterization in terms of *3-valued* models of revision programs. A *three-valued interpretation* is a pair of sets of atoms $\langle D_1, D_2 \rangle$ such that $D_1 \cap D_2 = \emptyset$. Consider a three-valued interpretation $V = \langle D_1, D_2 \rangle$. We say that V *3-satisfies* $\mathbf{in}(a)$ if $a \in D_1$. Similarly, V *3-satisfies* $\mathbf{out}(a)$ if $a \in D_2$. We say that V *3-satisfies* a revision rule c if V 3-satisfies the head of c whenever V 3-satisfies all literals in the body of c. Finally, V is a *three-valued model* of a revision program P if P 3-satisfies all rules in P. It is easy to show that a revision program is coherent if and only if P has a three-valued model.

We conclude this section by introducing another semantics for revision programs — the semantics of *supported revisions*. It is based on similar ideas as the semantics of supported models for logic programs [Cla78, MT93a].

Definition 3.11 A set of atoms $\mathcal{R}$ is a *P-supported* revision of $\mathcal{I}$ if $head(P_{\mathcal{R}})$ is coherent and $\mathcal{R} = \mathcal{I} \oplus head(P_{\mathcal{R}})$.

We will now present several properties of supported revisions. Our results generalize two well-known results on logic programming: (1) each supported model of a logic program P is a model of P, and (2) each stable model of a logic program P is a supported model of P.

Theorem 3.12 *Let P be a revision program and let $\mathcal{I}$ be a database. If a database $\mathcal{R}$ is a P-supported revision of $\mathcal{I}$ then $\mathcal{R}$ is a model of P.*

Proof: Clearly, $\mathcal{R} \models head(P_{\mathcal{R}})$ (Lemma 2.2). Consequently, $\mathcal{R} \models P_{\mathcal{R}}$. If $c \in P \setminus P_{\mathcal{R}}$, then $\mathcal{R} \not\models body(c)$ and, consequently, $\mathcal{R} \models c$. Hence, $\mathcal{R} \models P$. □

Theorem 3.13 *Let P be a revision program and let $\mathcal{I}$ be a database. If a database $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$, then $\mathcal{R}$ is a P-supported revision of $\mathcal{I}$.*

Proof: By Theorem 2.13, $head(P_{\mathcal{R}}) = NC(P_{\mathcal{I},\mathcal{R}})$. Hence, by the definition of P-justified revisions, $head(P_{\mathcal{R}})$ is coherent and $\mathcal{R} = \mathcal{I} \oplus head(P_{\mathcal{R}})$. □

4 Relation to logic programming

In Section 2 we proved that P-justified revisions can be defined similarly to stable models for logic programs [GL88]. We will now study the relationship between revision programming and logic programming in more detail. In particular, we will propose an interpretation of logic programs as revision programs.

Given a logic program clause c

$$p \leftarrow q_1, \ldots, q_m, \mathbf{not}(s_1), \ldots, \mathbf{not}(s_n) \tag{9}$$

we define the revision rule $rp(c)$ as

$$\mathbf{in}(p) \leftarrow \mathbf{in}(q_1), \ldots, \mathbf{in}(q_m), \mathbf{out}(s_1), \ldots, \mathbf{out}(s_n). \tag{10}$$

In addition, for a logic program P, we define the corresponding revision program $rp(P)$ by

$$rp(P) = \{rp(c) : c \in P\}. \tag{11}$$

Under this interpretation, several concepts in logic programming such as models, stable models and supported models of logic programs can faithfully be represented in terms of revision programs. (Recall that $\mathcal{M}$ is a supported model of a logic program P if $\mathcal{M} = head(P_{\mathcal{M}})$, where $P_{\mathcal{M}}$ is the set of those clauses in P whose bodies are satisfied by $\mathcal{M}$ [MT93a]).

Theorem 4.1 *Let P be a logic program.*

1. *A set of atoms $\mathcal{M}$ is a model of P if and only if $\mathcal{M}$ is a model of $rp(P)$.*

2. *A set of atoms $\mathcal{M}$ is a stable model of P if and only if $\mathcal{M}$ is an $rp(P)$-justified revision of $\emptyset$.*

3. *A set of atoms $\mathcal{M}$ is a supported model of P if and only if P if and only of $\mathcal{M}$ is an $rp(P)$-supported revision of $\emptyset$.*

Proof: (1) We leave to the reader proving this part of the assertion.
(2) First, notice that for every $\mathcal{R}$ the inertia $I(\emptyset, \mathcal{R})$ consists of negative literals only. Specifically,

$$I(\emptyset, \mathcal{R}) = \{\mathbf{out}(a) : a \notin \mathcal{R}\}.$$

Second, since the image of the logic program consists of in-rules only, the necessary change $NC(P_{\emptyset,\mathcal{M}})$ consists of positive literals only.

Now, let P be a logic program and $rp(P)$ its revision programming translation. Then $\mathcal{M}$ is a stable model of P if and only if $\mathcal{M}$ coincides with the least model of the Gelfond-Lifschitz reduct of P with respect to $\mathcal{M}$, $GL(P, \mathcal{M})$ (see [GL88]). Notice that $rp(P)_{\emptyset,\mathcal{M}}$ is obtained from $rp(P)$ by eliminating from the bodies of rules in $rp(P)$ all the literals in $I(\emptyset, \mathcal{M})$. But, as observed above, this inertia set consists of negative literals only. Since the reduced program consists of in-rules, we can now apply Lemma 2.5 and eliminate all rules which have negative literals in the body, since they will not be usable. It is easy to see that the resulting program is precisely the image of the original Gelfond-Lifschitz reduct[1] of P under the embedding rp. This implies that the necessary change of $rp(P)_{\emptyset,\mathcal{M}}$ is $\{\mathbf{in}(a) : a \in \mathcal{M}\}$. But then $\mathcal{M}$ is a P-justified revision of $\emptyset$.

It is easy to see that we used only equivalences, and so the converse implication holds as well.
(3) Since P is a logic program, it is easy to see that $\mathcal{M} = \emptyset \oplus \mathit{head}(rp(P)_{\mathcal{M}})$ if and only if $\mathcal{M} = \mathit{head}(P_{\mathcal{M}})$. This yields the assertion. □

Theorem 4.1 implies that every characterization of justified revisions has its counterpart — a characterization of stable models of logic programs. In particular, Theorem 2.12 implies a characterization of stable models in terms of a "symmetric" version of Gelfond-Lifschitz reduct. Similarly, Theorem 2.14 implies a characterization of stable models equivalent to the one provided in [BTK93] in terms of the assumption-based framework.

The second confirmation of evidence property (Theorem 3.3) together with the translation result (Theorem 4.1) imply the following confirmation property for stable models of logic programs.

Corollary 4.2 *Let P and P' be logic programs. Let $\mathcal{M}$ be a stable model of P. If $\mathcal{M} \models P'$ then $\mathcal{M}$ is a stable model of $P \cup P'$.*

[1] In Section 2 we used modified Gelfond-Lifschitz reduct.

Notice that under the assumptions of Corollary 4.2, although $\mathcal{M}$ remains the stable model of $P \cup P'$, the class of stable models of P is not, in general, preserved. That is, some of the stable models of P may no longer be stable models of $P \cup P'$, and new stable models of the larger program may appear.

We have just argued that logic programs can be regarded as special revision programs. In fact, the relationship between logic and revision programming is even more interesting. Przymusinski and Turner [PT95] discovered an encoding of revision programs in terms of logic programs which expresses justified revisions in terms of stable models. Thus, revision programs can be viewed as special logic programs. A natural question to ask is then: why to study revision programs at all?

In our view there are several reasons. The language of revision programming is tailored directly to situations in which we need to state and enforce constraints on presence and *absence* of elements in sets. Such features are important in the areas of database update and belief revision. Consequently, revision programming is a formalism which allows us to state problems of importance in these areas in an explicit and direct manner.

As shown by Przymusinski and Turner, revision programming can be embedded into logic programming with stable model semantics but, in the process, new symbols have to be introduced, the size of a program grows, justified revisions are not just stable models but have to be decoded from stable models, and finally, clear intuitions behind **in** and **out** operators become obscure. In addition, the embedding described in [PT95], while mapping justified revisions to stable models, does not map supported revisions to supported models, despite the existing natural correspondence between these two concepts, evident from the results presented in this section.

On the other hand, the embedding of logic programming into revision programming discussed in our paper is as simple as it can be. Up to a simple renaming of literals, it is an identity embedding. Consequently, results on revision programming directly and literally imply specifications that apply to logic programs. In particular, notions of models, supported models and stable models are uniformly mapped to the corresponding concepts in revision programming.

Next, as we discuss in more detail in Section 6, the existence of the encoding of revision programs as logic programs allows us to identify classes of logic programs with interesting arithmetic complexity properties. These programs and corresponding results are easy to describe in terms of revision programs while direct descriptions are less obvious.

Finally, from the vintage point of revision programs, it becomes clear that the realm of "programs" goes beyond just logic programs. There are programs which compute by *adding* new facts to the initially empty database (logic programs), programs that compute by *deleting* facts from a Herbrand base (revision programs dual to logic programs) and all combinations of these two cases.

5 Sequential revision process

Our definition of P-justified revisions has a certain "global" character. It is based on two operators that are applied to programs rather than to individual rules. The first of these operators assigns the reduct to a revision program, the other one assigns to the reduct the necessary change it implies. Hence, P-justified revisions of $\mathcal{I}$ can be viewed as the results of applying all rules of P to $\mathcal{I}$ "in parallel". In this section, we will present a different description of P-justified revisions. We will show that P-justified revisions of $\mathcal{I}$ are exactly those databases $\mathcal{R}$ which can be obtained from $\mathcal{I}$ by executing all rules of P *one by one* according to *some* enumeration of the rules in P. This property of the semantics of P-justified revisions is similar to the notion of *serializability* in transaction management.

For every rule $c \in P_{\mathcal{B}}$, $\mathcal{B} \models body(c)$. Hence, we will call all rules in $P_{\mathcal{B}}$ — *$\mathcal{B}$-applicable*. For example, the rule $\mathbf{in}(c) \leftarrow \mathbf{in}(a), \mathbf{out}(b)$ is not $\mathcal{B}$-applicable for $\mathcal{B} = \{a, b\}$ and it is $\mathcal{B}$-applicable for $\mathcal{B} = \{a, d\}$.

If a rule c is $\mathcal{B}$-applicable then its conclusion can be executed on the database $\mathcal{B}$ and, according to the head of c, an atom will be inserted to or deleted from $\mathcal{B}$. Assume that a certain well-ordering (enumeration) $\prec$ of the rules of P is given. Then, the following *sequential revision process* can be considered: in each step select the first rule according to $\prec$ which has not been selected before and which is applicable with respect to the *current* state of the database. Modify the database according to the head of the selected rule. Stop when selection of a rule, according to these principles, is no longer possible. The question that we deal with in this section is: how the results of such revision process relate to P-justified revisions?

Example 5.1 Let $\mathcal{B} = \emptyset$ and let P consist of the following two rules:

(1) $\mathbf{in}(c) \leftarrow \mathbf{out}(b)$ (2) $\mathbf{in}(b) \leftarrow \mathbf{in}(c)$.

Let us process the rules in the order they are listed. Rule (1) is applicable with respect to $\mathcal{B} = \emptyset$. Hence, the update $\mathbf{in}(c)$ is executed and we get a new database $\mathcal{B}_1 = \{c\}$. Now, the second rule is the first $\mathcal{B}_1$-applicable rule not applied yet. Hence, the update $\mathbf{in}(b)$ is executed. Consequently, the next database $\mathcal{B}_2 = \{b, c\}$ is obtained. Since there are no other rules left, the process stops. Notice, however, that rule (1) is not $\mathcal{B}_2$-applicable. Hence, the justification for inserting c is lost and $\mathcal{B}_2$ should not be regarded as a revision of $\mathcal{B}$. Observe that $\mathcal{B}_2$ is not a P-justified revision of $\mathcal{B}$.

Example 5.1 shows that there are cases when processing rules sequentially does not lead to a P-justified revision. The problem is that some of the rules applied at the beginning of the process may be rendered inapplicable by subsequent updates. But there is yet another source of problems.

Example 5.2 Let $\mathcal{B} = \{a\}$ and let P consist of the following three rules:

(1) $\mathbf{in}(c) \leftarrow \mathbf{out}(b)$ (2) $\mathbf{in}(d) \leftarrow \mathbf{in}(a)$ (3) $\mathbf{out}(c) \leftarrow \mathbf{in}(d)$.

Let us process the rules in the order they are listed. After using rule (1) we get a new database: $\mathcal{B}_1 = \{a, c\}$. Then, rule (2) is $\mathcal{B}_1$-applicable and after the update we obtain the database $\mathcal{B}_2 = \{a, c, d\}$. Finally, we apply rule (3) and produce the database $\mathcal{B}_3 = \{a, d\}$. Notice that all the rules applied in the process are $\mathcal{B}_3$-applicable. But $\mathcal{B}_3$ is not a model of the program P. The reason is that the set of literals produced in the process is not coherent. Hence, $\mathcal{B}_3$ cannot be regarded as a possible revised version of $\mathcal{B}$. Observe also that, since $\mathcal{B}_3$ is not a model of P it is not a P-justified revision of $\mathcal{B}$.

It turns out that Examples 5.1 and 5.2 capture all such cases when processing rules of the program according to some ordering does not yield a P-justified revision.

We will now formally define the *sequential revision process* and provide a precise formulation of the statement above. The approach we take is similar to our earlier result in which default extensions (and, hence, also stable models of logic programs) are characterized as results of some sequential computation by means of default rules (program clauses) [MT93a].

Let $\mathcal{I}$ be a set of atoms (a database) and let P be a revision program. Both $\mathcal{I}$ and P may be infinite. Let $\{c_t\}_{t<n}$ be an enumeration of rules in P. Here n stands for a natural number or an infinite ordinal (if the revision program P is infinite). Our argument is suitable for both finite and infinite case. For simplicity we assume that n is finite. A reader familiar with induction arguments will have no problem extending the argument to the transfinite case.

We define an integer n^*, a sequence of integers $\{t_q\}_{1 \leq q < n^*}$ and a sequence of sets $\{L_q\}_{q<n^*}$ as follows. First, we set

$$L_0 = \emptyset.$$

Let $p \geq 1$ be an integer. Assume that we have already defined *coherent* sets L_q of literals, for $q < p$, and integers t_q, for $1 \leq q < p$. Set

$$L_{<p} = \bigcup_{q<p} L_q.$$

Observe that $L_{<p}$ is coherent and define

$$\mathcal{B}_{<p} = \mathcal{I} \oplus L_{<p}.$$

Set $L_{<p}$ represents all updates produced by the process so far and $\mathcal{B}_{<p}$ is the result of revising $\mathcal{I}$ by $L_{<p}$. Next, define

$$A_{<p} = P_{\mathcal{B}_{<p}} \setminus \{c_{t_q} \colon 1 \leq q < p\}.$$

The set $A_{<p}$ consists of all rules that are applicable with respect to the database $\mathcal{B}_{<p}$ and have not been applied in the process yet.

If $A_{<p} = \emptyset$ then we stop the construction and set $n^* = p$. Otherwise, we define

$$t_p = \min\{t \colon c_t \in A_{<p}\}$$

and

$$L_p = L_{<p} \cup \{head(c_{t_p})\}.$$

If L_p is incoherent, define $n^* = p + 1$ and stop. Otherwise, continue. The cardinality argument ensures that the construction terminates.

After the construction terminates, define $L = \bigcup_{q<n^*} L_q$. If L is coherent, we also define $\mathcal{R} = \mathcal{I} \oplus L$. In such case, $\mathcal{R} = \mathcal{B}_{<n^*}$.

Note that n^* and the sequences $\{L_q\}_{q<n^*}$, and $\{t_q\}_{1\leq q<n^*}$ depend on the enumeration $\prec$ of P. We suppressed $\prec$ in the notation in order to simplify it.

The process described above is called the *sequential revision process* for the enumeration $\prec$ and a database $\mathcal{I}$. A well-ordering (enumeration) of a revision program P is called *a posteriori consistent* for $\mathcal{I}$ if all the rules of P that were applied in the corresponding sequential revision process ($\{c_{t_q}: 1 \leq q < n^*\}$) are applicable with respect to the resulting database $\mathcal{R}$. It is called *sound for a database* $\mathcal{I}$ if L is coherent. The ordering considered in Example 5.1 is not *a posteriori* consistent for $\mathcal{I} = \emptyset$, The ordering given in Example 5.2 is not sound for $\mathcal{I} = \{a\}$.

Theorem 5.3 *Let P be a revision program and let $\mathcal{I}$ be a database. A database $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if there exists an enumeration of P which is* a posteriori *consistent and sound for $\mathcal{I}$ and such that $\mathcal{R} = \mathcal{I} \oplus L$, where L is the set of literals produced by the corresponding sequential revision process.*

Proof: We will use in the proof the notation introduced in the definition of the sequential revision process. Let $\prec$ be an ordering of P, *a posteriori* consistent and sound for $\mathcal{I}$. Let L be the set of literals produced by the corresponding sequential revision process. It follows that $L = L_{<n^*}$. We will prove that the database $\mathcal{R} = \mathcal{I} \oplus L$ is a P-justified revision of $\mathcal{I}$.

Since the ordering $\prec$ is *a posteriori* consistent, it follows that for every q, $1 \leq q < n^*$, $\mathcal{R} \models body(c_{t_q})$. Hence,

$$\{c_{t_q}: 1 \leq q < n^*\} \subseteq P_{\mathcal{R}}.$$

Since $\mathcal{R} = \mathcal{B}_{<n^*}$, by the definition of the sequential revision process, $P_{\mathcal{R}} \setminus \{c_{t_q}: 1 \leq q < n^*\} = \emptyset$ (otherwise, the sequential revision process would not terminate on the integer n^*). Hence,

$$P_{\mathcal{R}} = \{c_{t_q}: 1 \leq q < n^*\}. \tag{12}$$

In particular,

$$L = head(P_{\mathcal{R}}). \tag{13}$$

Next, notice that for every p, $1 \leq p < n^*$,

$$body(c_{t_p}) \subseteq head(\{c_{t_q}: 1 < q < p\}) \cup I(\mathcal{I}, \mathcal{R}). \tag{14}$$

Indeed, let $\alpha \in body(c_{t_p})$. Then $\mathcal{B}_{<p} \models \alpha$. Since $c_{t_p} \in P_{\mathcal{R}}$, $\mathcal{R} \models \alpha$. If $\mathcal{I} \models \alpha$, then $\alpha \in I(\mathcal{B}_I, \mathcal{R})$ (Lemma 2.8). So, assume that $\mathcal{I} \not\models \alpha$, Since $\mathcal{B}_{<p} = \mathcal{I} \oplus L_{<p}$, by Lemma 2.2 it follows that $\alpha \in L_{<p}$. Since, $L_{<p} = \{head(c_{t_q}): 1 \leq q < p\}$, (14) follows.

Now, by (13), (14) and Lemma 2.6,

$$L = head(P_{\mathcal{R}}) = LM(P_{\mathcal{R}}|I(\mathcal{I},\mathcal{R})) = NC(P_{\mathcal{R}}|I(\mathcal{I},\mathcal{R})) = NC((P_{\mathcal{R}})_{\mathcal{I},\mathcal{R}}).$$

Hence, $\mathcal{R}$ is a $P_{\mathcal{R}}$-justified revision of $\mathcal{I}$ (recall that L is coherent). By Corollary 3.4, $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$.

Conversely, let us assume that $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$. Let $\{c_k\colon 1 \leq k < n\}$ be an enumeration of the rules of $P_{\mathcal{R}}$ such that for every $k \geq 1$,

$$body(c_k) \subseteq head(\{c_m\colon 1 < m < k\}) \cup I(\mathcal{I},\mathcal{R}).$$

Existence of such enumeration is guaranteed by Lemma 2.7. Indeed, by Theorem 2.13,

$$P_{\mathcal{R}}|I(\mathcal{I},\mathcal{R}) = P_{\mathcal{R}}|\mathcal{I} = (P_{\mathcal{I},\mathcal{R}})^u = (P|I(\mathcal{I},\mathcal{R}))^u.$$

Let $\prec_1$ be the enumeration of $P_{\mathcal{R}}$ consistent with the enumeration $\{c_k\colon 1 \leq k < n\}$ and let $\prec_2$ be any enumeration of $P \setminus P_{\mathcal{R}}$. For $c, c' \in P$, define $c \prec c'$ precisely in one of the following three cases:

1. $c \in P_{\mathcal{R}}$ and $c' \in P \setminus P_{\mathcal{R}}$
2. $c, c' \in P_{\mathcal{R}}$ and $c \prec_1 c'$
3. $c, c' \in P \setminus P_{\mathcal{R}}$ and $c \prec_2 c'$.

Clearly, $\prec$ is an enumeration of P. It is easy to show by induction that for every k, $1 \leq k < n$,

$$t_k = k.$$

Since $\mathcal{R} = \mathcal{B}_{<n}$, $A_n = \emptyset$. Hence, $n^* = n$ and the sequential revision process terminates with $\mathcal{B} = \mathcal{R}$. Consequently, $\prec$ is *a posteriori* consistent and sound. □

Theorem 5.3 states that P-justified revisions correspond to a class of orderings of the revision program P. It allows us to construct a P-justified revision of $\mathcal{I}$ by means of a process in which rules are applied sequentially one-by-one, assuming an *a posteriori* consistent and sound ordering of P can be found.

6 Safe programs

Given a database $\mathcal{B}$ and a revision program P, there is no guarantee that there is a database $\mathcal{B}'$ such that $\mathcal{B}'$ is a P-justified revision of $\mathcal{B}$. Moreover, if such revision exists, there is no guarantee that this revision is unique. This may be considered a drawback of revision programming as a proposal for the formalism to describe database revisions and updates. The goal of this section is to exhibit classes of revision programs which have a property that is highly desirable from the point of view of any practical database applications: to *every* initial database they assign a *unique* justified revision.

The problem outlined here appears also in other domains. For example, a logic program can have no, one or many stable models. This was of concern to the logic programming community and two important classes of logic programs were exhibited with exactly one stable model. These are the class of all Horn programs and the class of all stratified programs [ABW88]. We will now extend these concepts to the case of revision programming. We will first introduce the notion of a *safe program*. Safe programs generalize Horn programs to the domain of revision programming.

Let us observe that for any coherent set of literals L, the program $\{\alpha \leftarrow : \alpha \in L\}$ has the desired property that every initial database admits exactly one revision. The notion of safeness, introduced below, can be viewed as a generalization of the notion of a coherent set of literals.

Definition 6.1 A revision program P is safe if for every literal $\alpha \in \mathit{head}(P)$, $\alpha^D \notin \mathit{var}(P)$.

For example, the program

$$P_1 = \{\mathbf{in}(a) \leftarrow \mathbf{out}(b)\}$$

is safe. Similarly,

$$P_2 = \{\mathbf{in}(a) \leftarrow \mathbf{out}(b), \mathbf{in}(e), \mathbf{out}(c) \leftarrow \mathbf{out}(e), \mathbf{out}(d) \leftarrow \mathbf{in}(a), \mathbf{out}(b) \leftarrow\}$$

is also safe. However, the program

$$P_3 = \{\mathbf{in}(a) \leftarrow \mathbf{out}(b), \mathbf{in}(b) \leftarrow \mathbf{out}(a)\}$$

is not safe. In the context of logic programming safeness is the requirement that if the atom appears negated in the body of a logic program clause, then it does not appear among the heads of the clauses of the program. It is well known that each such logic program possesses a unique stable model.

Safeness is a syntactic condition and, more importantly, it can be checked in linear time. In addition, safe revision programs have several other desirable properties all essentially amounting to the fact that safe revision programs uniquely determine a revision of any initial database.

Theorem 6.2 *Let P be a safe revision program. Then, for every database $\mathcal{I}$:*

1. *There is a unique $\mathcal{R}$ such that $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$.*
2. *For every enumeration $\prec$ of P, the result of the sequential revision process for $\prec$ and $\mathcal{I}$ is the unique P-justified revision of $\mathcal{I}$.*
3. *The unique P-justified revision for $\mathcal{I}$ can be computed in time proportional to the total size of $\mathcal{I}$ and P.*

Proof: We will first prove (1). Let L be a set of literals such that $L \subseteq \mathit{head}(P)$. By safeness of P, L is coherent. Define $\mathcal{B}' = \mathcal{B} \oplus L$. We will first show that

$$P|\mathcal{B} = P_{\mathcal{B},\mathcal{B}'}. \tag{15}$$

Indeed, let α be a literal in a body of a rule $c \in P$. Assume that $\mathcal{B} \models \alpha$. Since $\alpha^D \notin L$ (safeness), $\mathcal{B}' \models \alpha$. Consequently, $\alpha \in I(\mathcal{B}, \mathcal{B}')$. Conversely, assume that $\alpha \in I(\mathcal{B}, \mathcal{B}')$. If $\alpha = \mathbf{in}(a)$, then $a \in \mathcal{B} \cap \mathcal{B}'$. If $\alpha = \mathbf{out}(a)$, then $a \notin \mathcal{B} \cup \mathcal{B}'$. In each case, $\mathcal{B} \models \alpha$. It follows that a literal is removed from the body of a rule in the construction of $P|\mathcal{B}$ if and only if it is removed during the construction of $P|I(\mathcal{B}, \mathcal{B}') = P_{\mathcal{B},\mathcal{B}'}$. Hence, (15) follows.

Let $\mathcal{D} = \mathcal{B} \oplus \mathit{NC}(P|\mathcal{B})$. Clearly, $\mathit{NC}(P|\mathcal{B})$ is coherent (by safeness of P). In addition, by (15),

$$\mathcal{D} = \mathcal{B} \oplus \mathit{NC}(P|\mathcal{B}) = \mathcal{B} \oplus \mathit{NC}(P_{\mathcal{B},\mathcal{D}}).$$

Hence, $\mathcal{D}$ is a P-justified revision of $\mathcal{B}$. Uniqueness of $\mathcal{D}$ follows directly from (15). Indeed, if $\mathcal{D}$ and $\mathcal{D}'$ are P-justified revisions of $\mathcal{B}$, then

$$\mathcal{D} = \mathcal{B} \oplus \mathit{NC}(P_{\mathcal{B},\mathcal{D}}) = \mathcal{B} \oplus \mathit{NC}(P|\mathcal{B}) = \mathcal{B} \oplus \mathit{NC}(P_{\mathcal{B},\mathcal{D}'}) = \mathcal{D}'.$$

(2) Observe that safeness implies that no rule has in its body a literal whose dual appears in $\mathit{head}(P)$. Consequently, when the sequential revision process terminates, all the rules that were applied in the process, remain applicable with respect to the resulting database. In other words, $\prec$ is *a posteriori* consistent. Moreover, since $\mathit{head}(P)$ is coherent, $\prec$ is sound. Hence, for every enumeration $\prec$, the result of the sequential revision process is a P-justified revision of $\mathcal{B}$ (and it is unique by (1)).

(3) Notice the following facts. First, the reduction process of P with respect to $\mathcal{B}$ can be performed in time proportional to the total size of P and $\mathcal{B}$. Next, we find the least model of reduced program. This can be done in time proportional to its size (see [DG84]), which is bound by the size of the original program. Finally, the database $\mathcal{B}$ has to be updated by the computed set of literals (necessary change). This again can be accomplished in time proportional to the total size of $\mathcal{B}$ and P. □

Property (1) in Theorem 6.2 generalizes a well-known property of Horn programs that states that every Horn program has a unique least model. Property (3) and its proof imply a deterministic, linear-time algorithm for computing justified revisions for *safe* programs.

As in logic programming, some of useful properties of safe programs can be extended to a wider class of programs.

Definition 6.3 Let P be a revision program and let $\{P_t\}_{0<t<n}$ be a partition of P. We say that $\{P_t\}_{0<t<n}$ is a *stratification* of P if for every $0 < t < n$:

1. P_t is safe, and

2. if $\alpha \in head(P_t)$ then $\alpha, \alpha^D \notin \bigcup_{q<t} var(P_q)$.

Clearly, each safe program is stratified. Notice also that revision programs obtained from locally stratified logic programs under the interpretation described in Section 2 are stratified according to Definition 6.3.

To test if a finite revision program P is stratified and, if so, to find a partition of P into strata, one can use a modified version of the algorithm of Apt, Blair and Walker [ABW88]. It takes linear time in the size of a revision program P.

Several important properties of safe revision programs can be extended to the class of stratified programs. In particular, we have the following generalization of Theorem 6.2(1).

Theorem 6.4 *Let P be a stratified revision program. For every database $\mathcal{B}$ there exists a unique database $\mathcal{D}$ such that $\mathcal{D}$ is a P-justified revision of $\mathcal{B}$.*

Proof (sketch): Let $\{P_t\}_{0<t<n}$ be a stratification of P. For every t, $0 < t < n$, define

$$\mathcal{B}_t = \{a \in \mathcal{B} : \mathbf{in}(a) \in head(P_t) \text{ or } \mathbf{out}(a) \in head(P_t)\}.$$

Intuitively, $\mathcal{B}_t$ is this part of $\mathcal{B}$ that can be affected by revisions implied by the program P_t. Next, define $\mathcal{B}_0 = \mathcal{B} \setminus \bigcup_{0<t<n} \mathcal{B}_t$. Clearly, $\{\mathcal{B}_t\}_{t<n}$ is a partition of $\mathcal{B}$.

Now, we proceed as follows. First, we define $\mathcal{D}_0 = \mathcal{B}_0$, and we put $\mathcal{D}_t$ to be a unique P_t-justified revision of the database $\mathcal{B}_t \cup \mathcal{D}^{<t}$, where $\mathcal{D}^{<t} = \bigcup_{r<t} \mathcal{D}_r$. Then, we define

$$\mathcal{D} = \bigcup_{t<n} \mathcal{D}_t.$$

Here is an informal account of what happens during the construction. Since $\mathcal{B}_0$ cannot be revised by means of P at all, it is put into $\mathcal{D}$ at once. Subsequently, at each stage t we revise $\mathcal{B}_t \cup \mathcal{D}^{<t}$. Observe that $\mathcal{D}^{<t}$ is the result of revisions at earlier stages. Due to stratification of P, the program P_t cannot modify $\mathcal{D}^{<t}$. Hence, it will remain unchanged. What will change is $\mathcal{B}_t$. However, since the rules of P_t may contain in their bodies literals whose status is established in earlier stages of the construction, $\mathcal{D}^{<t}$ must be explicitly used as input. At the end we output the union of constructed layers.

We need to show that $\mathcal{B} \oplus NC(P_{\mathcal{B},\mathcal{D}}) = \mathcal{D}$. To this end, we observe (the proof is left to the reader) that by stratification

$$NC(P_{\mathcal{B},\mathcal{D}}) = \bigcup_{0<t<n} NC(P_t|(\mathcal{B}_t \cup \mathcal{D}^{<t})).$$

Hence

$$\mathcal{B} \oplus NC(P_{\mathcal{B},\mathcal{D}}) = \left(\bigcup_{t<n} \mathcal{B}_t\right) \oplus \left(\bigcup_{0<t<n} NC(P_t|(\mathcal{B}_t \cup \mathcal{D}^{<t}))\right).$$

It is easy to see that the latter set coincides with

$$\mathcal{B}_0 \cup \left(\bigcup_{0<t<n} \mathcal{B}_t \oplus NC(P_t|(\mathcal{B}_t \cup \mathcal{D}^{<t}))\right) = \mathcal{D}_0 \cup \bigcup_{0<t<n} \mathcal{D}_t = \mathcal{D}.$$

Hence, the existence of a P-justified revision of $\mathcal{B}$ follows.

The uniqueness part follows the usual line of stratification arguments, see [MT93a]. We leave the task of checking this to the reader. □

Let us consider a stratification $\{P_t\}_{0<t<n}$ of a stratified program P. An enumeration $\prec$ of P *agrees* with the the stratification $\{P_t\}_{0<t<n}$ if for every $t_1 < t_2 < n$, and for every rules $c_1 \in P_{t_1}$ and $c_2 \in P_{t_2}$, $c_1 \prec c_2$. It is easy to see that such orderings exist. Now, we can generalize Theorem 6.2(2).

Theorem 6.5 *Let P be a stratified revision program and let $\mathcal{I}$ be a database. Then for every stratification $\{P_t\}_{0<t<n}$ of P and for every enumeration $\prec$ of P which agrees with the stratification $\{P_t\}_{0<t<n}$, the result of the sequential revision process for $\prec$ and $\mathcal{I}$ is the unique P-justified revision of $\mathcal{I}$.* □

It should be clear that the argument of Theorem 6.4 yields an algorithm for computation of the unique P-justified revision of database $\mathcal{B}$ whenever P is stratified. The algorithm computes the revision in stages and, in each stage, a different stratum P_t is used. Since P_t is safe, the revision can be computed in time linear in the total the size of P_t, $\mathcal{B}_t$ and $\mathcal{D}^{<t}$. If we maintain the set of literals from $\mathcal{D}$ computed up to stage t (that is, the set $\mathcal{D}^{<t}$) as a characteristic array, then the computation of $P_t|(\mathcal{B}_t \cup \mathcal{D}^{<t})$ can be performed in time linear in the size of P_t and $\mathcal{B}_t$. Consequently, the computation of a unique revision of a finite database $\mathcal{B}$ by a finite stratified revision program P can be accomplished in time linear in the total size of P and $\mathcal{B}$. Hence, we have the following generalization of Theorem 6.2(3).

Theorem 6.6 *Let P be a finite stratified revision program and let $\mathcal{I}$ be a finite database. Then a unique P-justified revision of $\mathcal{I}$ can be computed in time proportional to the total size of P and $\mathcal{I}$. In particular, the assertion holds for safe programs.* □

We will conclude this section with a discussion of complexity issues for infinite safe programs. Apt and Blair [AB90] proved that finite stratified predicate programs and infinite recursive propositional programs with at most n strata compute precisely Σ^0_n sets of natural numbers (for $n = 1$ this result was proved by Smullyan [Smu68]; see also [AN78]). This result provides an insight in the relationship between the complexity of stratified logic programs, measured in terms of the number of strata, and the complexity of sets that these programs compute. Revision programming allows for a subtler study by explicitly allowing for deletions and by providing *two* control parameters: the complexity of an initial database and the complexity of a revision program (expressed in the number of strata). We will illustrate this thesis with one example, in which the complexity of justified revisions of recursive databases by means of recursive safe programs is determined.

Definition 6.7 [EHK81]

1. A subset $A \subseteq \omega$ is called a *d.r.e. set* (difference of r.e. sets) if there are r.e. sets B, C such that $A = B \setminus C$.

2. A subset $A \subseteq \omega$ is *weakly d.r.e.* if both A and $\omega \setminus A$ are d.r.e. sets.

The class of d.r.e. sets is not closed under complements in the very same way as r.e. sets are not closed under complements. However, the class of weakly d.r.e. sets is closed under complement. In some sense, weakly d.r.e sets play the role of "recursive" sets with respect to d.r.e. sets. More on d.r.e. sets can be found in [EHK81]. The relationship of weakly d.r.e. sets to revision programming is explained in the following theorem.

Theorem 6.8 *Let $\mathcal{B}$ be a recursive database and P be a recursive safe program. Then the result of P-justified revision of $\mathcal{B}$ is a weakly d.r.e. set.*

Proof: Let $\mathcal{B}$ be a recursive database and P a recursive and safe program. First, observe that $P|\mathcal{B}$ is recursively enumerable. Indeed, let P be a range of a recursive function f. We define a function g as follows. On input n, function g first computes the rule $f(n) = \alpha \leftarrow \beta_1, \ldots, \beta_k$. When this is done, function g checks if for all j, $1 \leq j \leq k$, $\mathcal{B} \models \beta_j$ (recall that $\mathcal{B}$ is recursive). If so, β_j is eliminated from the body of $f(n)$, otherwise it is left there. Clearly, the function g so defined is recursive and its range is $P|\mathcal{B}$. Therefore $P|\mathcal{B}$ is recursively enumerable.

Next, notice that since $P|\mathcal{B}$ is recursively enumerable, so is its least model $NC(P|\mathcal{B})$. This in turn implies that $NC^+(P|\mathcal{B})$ and $NC^-(P|\mathcal{B})$ are recursively enumerable. Moreover, P being safe, $NC(P|\mathcal{B})$ is guaranteed to be coherent. It is easy to see that a unique P-justified revision $\mathcal{B}'$ of $\mathcal{B}$ can be written as:

$$\mathcal{B}' = (\mathcal{B} \cup \{a : \mathbf{in}(a) \in NC^+(P|\mathcal{B})\}) \setminus \{b : \mathbf{out}(b) \in NC^-(P|\mathcal{B})\}.$$

Since, $\{a : \mathbf{in}(a) \in NC^+(P|\mathcal{B})\}$ and $\{b : \mathbf{out}(b) \in NC^-(P|\mathcal{B})\}$ are recursively enumerable and $\mathcal{B}$ is recursive, we see that $\mathcal{B}'$ is a d.r.e. set.

Finally, notice that since $\mathcal{B}$ is recursive then so is $\overline{\mathcal{B}}$. Similarly, since P is recursive then so is P^D (recall that P^D stands for the dual program for P, described in the proof of Theorem 3.8). In addition, it is easy to see that if a revision program P is safe, P^D is safe, too. Consequently, by the Duality Theorem (Theorem 3.8) and the first part of our argument, $\overline{\mathcal{B}'}$ is also a d.r.e. set. Thus, $\mathcal{B}'$ is a weakly d.r.e. set, as claimed. □

Let us discuss the role of Theorem 6.8. It implies that recursive safe revision programs on recursive inputs compute strictly less than stratified logic programs with two strata. Indeed, it is well known [EHK81] that the class of weakly d.r.e. sets is strictly smaller than the class Δ_2^0 and, consequently, than the class Σ_2^0, which is computed by stratified programs with two strata. In the same time, recursive safe revision programs on recursive inputs compute strictly more than Horn programs. For, instance, it is easy to construct recursive safe revision programs computing co-r.e. sets on recursive inputs. Hence, revision programs give rise to a finer classification of logic programs with respect to their expressibility.

7 Complexity and algorithms

We will now study the complexity of problems involving justified revisions. Related results concerning logic programming with stable and supported models can be found in [MT91, Sch95]. We will also present algorithms for computing justified revisions, given a finite revision program and a finite initial database.

Problems we are interested in can be be grouped into three broad categories:

Existence: Does a justified revision exist?

Membership_in_some: Does an atom a belong to some justified revision?

Membership_in_all: Does an atom a belong to all justified revisions?

To study the complexity of these problems we will need simple auxiliary facts.

Theorem 7.1 *Let P be a revision program.*

1. *There exist databases $\mathcal{I}$ and $\mathcal{R}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$ if and only if P has a model.*
2. *If a database $\mathcal{R}$ is a P-justified revision of a database $\mathcal{I}$ then there is a coherent set of literals $L \subseteq head(P)$ such that $\mathcal{R} = \mathcal{I} \oplus L$.*

Proof: If $\mathcal{R} \in JR_P(\mathcal{I})$, then $\mathcal{R}$ is a model of P (Theorem 3.1). Conversely, if $\mathcal{B}$ is a model of P, then $\mathcal{B} \in JR_P(\mathcal{B})$ (Theorem 3.5). The second part of the theorem is a direct consequence of the definition of P-justified revisions. □

Let us also observe that the following algorithm **Check** correctly verifies whether a database $\mathcal{R}$ is a P-justified revision of a database $\mathcal{I}$.

Check$(P, \mathcal{I}, \mathcal{R})$
(1) Compute the program $P_{\mathcal{I},\mathcal{R}}$
(2) Compute $NC(P_{\mathcal{I},\mathcal{R}})$
(3) **if** $NC(P_{\mathcal{I},\mathcal{R}})$ is incoherent **then return**{**false**}
(4) **if** $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$ **then return**{**true**} **else return**{**false**}

It is clear that algorithm **Check** can be implemented to run in polynomial time (in fact, a linear-time implementation is also possible) in the size of P, $\mathcal{I}$ and $\mathcal{R}$.

We are ready to investigate the complexity of decision problems associated with revision programming. We will consider several versions and specializations of the three broad problems, Existence, Membership_in_some and Membership_in_all, that are mentioned above. They are described in Table 7.1. In this table, P stands for a finite revision program, α for a literal, a for an atom, and $\mathcal{I}$ and $\mathcal{R}$ for finite databases.

Problem	Input	Question
E1	P	?$\exists$ $\mathcal{I}$, $\mathcal{R}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$
E2	P, $\mathcal{I}$	?$\exists$ $\mathcal{R}$ such that $\mathcal{R}$ $\mathcal{R} \in JR_P(\mathcal{I})$
E3	P, $\mathcal{R}$	?$\exists$ $\mathcal{I}$ such that $\mathcal{R}$ $\mathcal{R} \in JR_P(\mathcal{I})$
MS1	P, α	?$\exists$ $\mathcal{I}$, $\mathcal{R}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$ and $\alpha \in I(\mathcal{I}, \mathcal{R})$
MS2	P, α	?$\exists$ $\mathcal{I}$, $\mathcal{R}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$ and $\alpha \notin I(\mathcal{I}, \mathcal{R})$
MS3	P, a, $\mathcal{I}$	?$\exists$ $\mathcal{R}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$ and $a \in \mathcal{R}$
MS4	P, a, $\mathcal{R}$	?$\exists$ $\mathcal{I}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$ and $a \in \mathcal{I}$
MS5	P, a, $\mathcal{I}$	?$\exists$ $\mathcal{R}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$ and $a \notin \mathcal{R}$
MS6	P, a, $\mathcal{R}$	?$\exists$ $\mathcal{I}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$ and $a \notin \mathcal{I}$
MA1	P, α	?$\forall$ $\mathcal{I}$, $\mathcal{R}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$, $\alpha \notin I(\mathcal{I}, \mathcal{R})$
MA2	P, α	?$\forall$ $\mathcal{I}$, $\mathcal{R}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$, $\alpha \in I(\mathcal{I}, \mathcal{R})$
MA3	P, a, $\mathcal{I}$	?$\forall$ $\mathcal{R}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$, $a \notin \mathcal{R}$
MA4	P, a, $\mathcal{R}$	?$\forall$ $\mathcal{I}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$, $a \in \mathcal{I}$
MS5	P, a, $\mathcal{I}$	?$\forall$ $\mathcal{R}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$, $a \in \mathcal{R}$
MS6	P, a, $\mathcal{R}$	?$\forall$ $\mathcal{I}$ such that $\mathcal{R} \in JR_P(\mathcal{I})$ and $a \in \mathcal{I}$

Table 7.1 Decision problems in revision programming

For these problems, we have the following result.

Theorem 7.2 *(1) Problems* E1 *and* E2 *are NP-complete. Problem* E3 *can be decided in time linear in the size of P and $\mathcal{R}$.*
(2) Problems MS1 - MS3 *and* MS5 *are NP-complete. Problems* MS4 *and* MS6 *are in P.*
(3) Problems MA1 - MA3 *and* MA5 *are coNP-complete. Problems* MA4 *and* MA6 *are in P.*

Proof: (1) Consider a nondeterministic algorithm that, given P, first guesses a database $\mathcal{B}$ consisting of some atoms occurring in P and, then, checks whether $\mathcal{B}$ is a model of P. This last task can be accomplished in polynomial time. By Theorem 7.1(1), problem E1 is in NP. We will now show that E1 is NP-complete by describing a polynomial-time reduction of the propositional satisfiability problem to E1. Let $\mathcal{C} = \{c_1, \ldots, c_k\}$ be a collection of clauses. Assume that each clause is in the form

$$a_1 \wedge a_2 \wedge \ldots \wedge a_k \rightarrow a_{k+1},$$

where each a_i is a literal. For each such clause C define a revision rule $rp(c)$

$$\alpha_{k+1} \leftarrow \alpha_1, \alpha_2, \ldots, \alpha_k,$$

where $\alpha_i = \mathbf{in}(a_i)$, if a_i is an atom, and $\alpha_i = \mathbf{out}(a_i')$, if a_i is the negation of an atom a_i'. It is easy to see that $\mathcal{B}$ is a model of $\mathcal{C}$ if and only if $\mathcal{B}$ is a model of a revision program $\{rp(c)\colon c \in \mathcal{C}\}$. Hence, by Theorem 7.1(1), $\mathcal{C}$ is satisfiable if and only if problem E1 has answer YES for the revision program $\{rp(c)\colon c \in \mathcal{C}\}$.

By Theorem 7.1 it follows that the problem E2 is in NP. Indeed, to decide (nondeterministically) whether there is a P-justified revision of a database $\mathcal{I}$, it is enough to guess a subset L of $head(P)$, check that it is coherent, compute $\mathcal{R} = \mathcal{I} \oplus L$ and, finally, use algorithm **Check**$(P, \mathcal{I}, \mathcal{R})$ to verify that $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$.

Furthermore, problem E2 is NP-complete. It follows from the observation that under the restriction to programs consisting of in-rules only and to the case $\mathcal{I} = \emptyset$, problem E2 becomes equivalent to the question whether a logic program has a stable model (Theorem 4.1), which is known to be NP-complete [MT91].

Finally, problem E3 is equivalent to the problem whether $\mathcal{R}$ is a model of P. Hence, E3 can be decided in linear time.

(2) We will start with problem MS4. Consider the following algorithm.

1. If $\mathcal{R}$ is not a model of P then return NO and stop.
2. If $\mathcal{R}$ is a model of P and $a \in \mathcal{R}$ then return YES and stop.
3. if $\mathcal{R}$ is a model of P and $a \notin \mathcal{R}$ then, if $\mathcal{R}$ is a P-justified revision of $\mathcal{R} \cup \{a\}$ then return YES and stop, otherwise, return NO and stop.

This algorithm can be implemented to run in polynomial time (using algorithm **Check** described earlier). It is also correct. Indeed, if $\mathcal{R}$ is not a model of P there is no $\mathcal{I}$ such that $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ (Theorem 3.1). If $\mathcal{R}$ is a model of P and $a \in \mathcal{R}$, the answer is YES since $\mathcal{R}$ is a P-justified revision of $\mathcal{R}$ (Theorem 3.5). Finally, if $\mathcal{R}$ is a model of P and $a \notin \mathcal{R}$ then, by Theorem 3.2, there is a database $\mathcal{I}$ such that $a \in \mathcal{I}$ and $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $\mathcal{R}$ is a P-justified revision of $\mathcal{R} \cup \{a\}$. It follows that MS4 is in P.

In a similar way, one can show that MS6 is in P. The key observation (again implied by Theorem 3.2) is that if $\mathcal{R}$ is a model of P and $a \in \mathcal{R}$, then there exists a database $\mathcal{I}$ such that $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ and $a \notin \mathcal{I}$ if and only if $\mathcal{R}$ is a P-justified revision of $\mathcal{R} \setminus \{a\}$.

Next, we will deal with problems MS1, MS2, MS3 and MS5, All of them are in NP. For example, an algorithm to decide MS1 first nondeterministically guesses two databases $\mathcal{I}$ an $\mathcal{R}$. Then, it checks that $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ (using algorithm **Check**). Finally, it checks that $\alpha \in I(\mathcal{I}, \mathcal{R})$. It is clear that this nondeterministic algorithm runs in polynomial time.

NP-completeness of MS3 and MS5 follows from the fact that their restricted versions (when P consists of in-rules only and $\mathcal{I} = \emptyset$) are equivalent to the problems to decide whether a given element belongs (does not belong, respectively) to a stable model of a logic program, which is known to be NP-complete ([MT91, Sch95]).

Let us consider now problem MS1. It is easy to see that problem E1 can be polynomially reduced to MS1. Let P be a finite revision program. Let x be an atom not in U. Define $P' = P \cup \{\mathbf{in}(x) \leftarrow \mathbf{in}(x)\}$. It is easy to see that $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $\mathcal{R} \cup \{x\}$ is a P'-justified revision of $\mathcal{I} \cup \{x\}$. Hence, any algorithm for MS1, when used for P' and x, decides problem E1. It follows that MS1 is NP-complete.

A similar argument works for problem MS2. As before, let P be a finite revision program and let x be an atom not in U. Define $P' = P \cup \{\mathbf{in}(x) \leftarrow\}$. One can now show that $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $\mathcal{R} \cup \{x\}$ is a P'-justified revision of $\mathcal{I}$. Hence, E1 can be polynomially reduced to MS2, which implies that MS2 is NP-complete.

(3) Observe that problem MAi is the complement of problem MSi, $1 \leq i \leq 6$. Consequently, the result follows from Theorem 7.2. □

We will conclude this section with two straightforward algorithms for computing all P-justified revisions for a given database $\mathcal{I}$. The first of these algorithms, **Guess_and_Check**, is based directly on the definition of justified revisions and on Theorem 7.1.

Guess_and_Check$(P, \mathcal{I})$
(1) **for** every coherent subset L of $head(P)$ **do**
(2) $\quad \mathcal{B} := \mathcal{I} \oplus L$
(3) $\quad$ **if Check**$(P, \mathcal{I}, \mathcal{B})$ **then** output $\mathcal{B}$ as a P-justified revision of $\mathcal{I}$.

The next algorithm is based on the sequential revision process idea. Namely, it is based on Theorem 5.3 which states that all P-justified revisions of $\mathcal{I}$ can be found if all possible orderings of rules in P are considered. In the description given below, L stands for the set of literals produced so far in the construction, $\mathcal{B}$ stands for the current database, R consists of all the rules that were already used and A stands for the rules that can be applied in a current stage. If the algorithm does not generate any output, $\mathcal{I}$ has no P-justified revisions.

Sequential_Revision_Process$(P, \mathcal{I})$
(1) **for** all total orderings $\prec$ of P **do**
(2) $\quad L := \emptyset$
(3) $\quad \mathcal{B} := \mathcal{I}$
(4) $\quad R := \emptyset$
(5) $\quad A := P_{\mathcal{B}}$
(6) $\quad$ **while** L is coherent **and** $A \neq \emptyset$ **do**
(7) $\quad\quad c$:= $\prec$-first rule in A
(8) $\quad\quad L := L \cup \{head(c)\}$
(9) $\quad\quad R := R \cup \{c\}$
(10) $\quad\quad$ **if** L is coherent **then**
(11) $\quad\quad\quad \mathcal{B} := \mathcal{I} \oplus L$
(12) $\quad\quad\quad A := P_{\mathcal{B}} \setminus R$
(13) $\quad$ **if** L is coherent **and** $P_{\mathcal{B}} = R$ **then** report "$\mathcal{B}$ is a P-justified revision of $\mathcal{I}$"

Checking coherence of L in line (13) verifies that $\prec$ is sound, and checking that $P_{\mathcal{B}} = R$ decides *a posteriori* consistency.

As stated, this algorithm is more complex than the previous one (the main loop has to

be repeated $|P|!$ times). However, it can be improved. In fact, to insure its completeness, it is enough to consider only a subset of the set of all orderings of cardinality at most $2^{|P|}$.

These algorithms can only be regarded as the departure point for any serious study of algorithms for computing justified revisions. Specifically, as in the case of stable model computation, search space pruning techniques have to be developed to make these algorithms practical. This is the subject of a work in progress.

High complexity of computing justified revisions is a serious problem. Fortunately, there are wide classes of programs (safe and stratified) whose computational properties are much better. We have discussed them in Section 6.

8 Conclusions

In this paper we introduced revision programming — a logic-based framework for describing constraints on databases and providing a computational mechanism to enforce them.

Revision programming has an elegant theory. The change (revisions performed) is minimal and justified by the revision program based on the inertia set — a collection of literals that do not change status during the revision. There is a natural notion of duality, which allows us to treat positive and negative literals uniformly. Complexity of reasoning with revision programs is well understood and algorithms to compute justified revisions are known. In general, a revision program does not guarantee a unique revision for every initial database. However, we found two wide classes of logic programs which do have this desirable property.

Revision programming is closely related to logic programming. There is a simple embedding of logic programs in revision programming under which such concepts as model, stable model and supported model of a program are preserved. Looking at logic programming from the perspective of revision programming explains why positive and negative literals cannot be treated as dual notions in logic programming. In the same time, there are recent results that show that revision programs can be embedded in logic programs [PT95].

Several important questions remain open. First, connections with logic programming have to be further explored, especially, a possibility of developing a revision programming version of well-founded semantics. Another interesting avenue of research is to study the exact relationship of revision programming to theories of update and belief revision by Alchourrón, Gärdenfors and Makinson [AGM85], and Katsuno and Mendelzon [KM91].

Acknowledgments

This work was partially supported by the National Science Foundation under the grant IRI-9400568.

References

[AV91] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43:62–124, 1991.

[AGM85] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[AN78] H. Andreka and I. Nemeti. The generalized completeness of Horn predicate logic as a programming language. *Acta Cybernetica*, 4:3–10, 1978.

[Apt90] K. Apt. Logic programming. In J. van Leeuven, editor, *Handbook of theoretical computer science*, pages 493–574. MIT Press, Cambridge, MA, 1990.

[AB90] K. Apt and H.A. Blair, Arithmetical classification of perfect models of stratified programs. *Fundamenta Informaticae*, 12:1–17, 1990.

[ABW88] K. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of deductive databases and logic programming*, pages 89–142, Los Altos, CA, 1988. Morgan Kaufmann.

[BF91] N. Bidoit and C. Froidevaux. Negation by default and unstratifiable logic programs. *Theoretical Computer Science*, 78:85–112, 1991.

[BTK93] A. Bondarenko, F. Toni and R.A. Kowalski. An assumption-based framework for non-monotonic reasoning. In A. Nerode and L. Pereira, editors, *Logic programming and non-monotonic reasoning. Proceedings of the Second International Workshop*, pages 171–189. MIT Press, 1993.

[Cla78] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and data bases*, pages 293–322. Plenum Press, 1978.

[DG84] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.

[EHK81] R.L. Epstein, R. Haas, and R.L. Kramer. Hierarchies of sets and degrees below $0'$. In M. Lerman, J.H. Schmerl, and R.I. Soare, editors, *Logic Year 1979-80*, pages 32–48. Springer Verlag, 1981. S.L.N. in Mathematics 859.

[GL88] M. Gelfond and V. Lifschitz. The stable semantics for logic programs. In R. Kowalski and K. Bowen, editors, *Proceedings of the 5th international symposium on logic programming*, pages 1070–1080, Cambridge, MA, 1988. MIT Press.

[KM91] H. Katsuno and A.O. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence Journal*, 52:263 – 294, 1991.

[Llo84] J. Lloyd. *Foundations of logic programming*. Berlin: Springer-Verlag, 1984.

[MT89] W. Marek and M. Truszczyński. Stable semantics for logic programs and default theories. In E.Lusk and R. Overbeek, editors, *Proceedings of the North American conference on logic programming*, pages 243–256, Cambridge, MA, 1989. MIT Press.

[MT91] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38:588–619, 1991.

[MT93a] W. Marek and M. Truszczyński. *Nonmonotonic logics; context-dependent reasoning*. Berlin: Springer-Verlag, 1993.

[MT93b] W. Marek and M. Truszczyński. Reflexive autoepistemic logic and logic programming. In A. Nerode and L. Pereira, editors, *Logic programming and non-monotonic reasoning*. MIT Press, 1993.

[MT94] W. Marek and M. Truszczyński. Revision specifications by means of revision programs. In *Logics in AI. Proceedings of JELIA '94*. Lecture Notes in Artificial Intelligence. Springer-Verlag, 1994.

[MT95] W. Marek and M. Truszczyński. Revision programming, database updates and integrity constraints. In *Proceedings of the 5th International Conference on Database Theory — ICDT 95*, pages 368–382. Berlin: Springer-Verlag, 1995. Lecture Notes in Computer Science 893.

[PT95] T.C. Przymusiński and H. Turner. Update by means of inference rules. In *Proceedings of LPNMR'95*, pages 156–174. Berlin: Springer-Verlag, 1995. Lecture Notes in Computer Science 928.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Sch95] J. Schlipf. The expressive powers of the logic programming semantics. *Journal of the Computer Systems and Science*, 51:64 – 86, 1995.

[Sch92] G.F. Schwarz. Minimal model semantics for nonmonotonic modal logics. In *Proceedings of LICS-92*, 1992.

[Smu68] R.M. Smullyan. *First-order logic*. Berlin: Springer-Verlag, 1968.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, Rockville, MD, 1988.

[vEK76] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.