Revision programming = logic programming + integrity constraints

Victor Marek, Inna Pivkina, and Mirosław Truszczyński

Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046
marek|inna|mirek@cs.engr.uky.edu

Abstract. We study *revision programming*, a logic-based mechanism for enforcing constraints on databases. The central concept of this approach is that of a *justified revision based on a revision program*. We show that for any program P and for any pair of initial databases $\mathcal{I}$ and $\mathcal{I}'$ we can transform (shift) the program P to a program P' so that the size of the resulting program does not increase and so that P-justified revisions of $\mathcal{I}$ are shifted to P'-justified revisions of $\mathcal{I}'$. Using this result we show that revision programming is closely related to a subsystem of general logic programming of Lifschitz and Woo. This, in turn, allows us to reduce revision programming to logic programming extended by the concept of a constraint with a suitably modified stable model semantics. Finally, we use the connection between revision programming and general logic programming to introduce a disjunctive version of our formalism.

1 Introduction

Revision programming was introduced in [MT98] as a formalism to describe and study the process of database updates. In this formalism, the user specifies updates by means of *revision rules*, that is, expressions of the following two types:

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

or

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

where a, a_i and b_i are data items from some finite universe, say U. Rules of the first type are called *in-rules* and rules of the second type are called *out-rules*.

Revision rules have a declarative interpretation as constraints on databases. For instance, an in-rule (1) imposes on a database the following condition: a is *in* the database, or at least one a_i, $1 \leq i \leq m$, is *not* in the database, or at least one b_j, $1 \leq j \leq n$, is *in* the database.

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

In [MT98], a precise semantics for *revision programs* (collections of revision rules) was defined. Given a revision program P and a database $\mathcal{I}$, this semantics specifies a family of databases, each of which might be chosen as an update of $\mathcal{I}$ by means of the program P. These revised databases are called P-justified revisions of $\mathcal{I}$. In [MT98] (and in the earlier papers [MT94] and [MT95]), basic properties of justified revisions were established. Subsequently, revision programming was studied in the context of situation calculus [Bar97] and reasoning about actions [McCT95,Tur97].

Revision programming has also been investigated from the perspective of its close relationship with logic programming. In [MT98], it was argued that revision programming extends logic programming with stable semantics by showing that revision programs consisting of in-rules only can be identified with logic programs. A converse embedding — an encoding of revision programs as logic programs — was constructed in [PT97]. The techniques from this paper are now being exploited in the study of the problem of updating logic programs [AP97] and resulted in a new paradigm of dynamic logic programming [ALP+98]. Well-founded semantics for a formalism closely related to revision programming was discussed in [BM97].

The key property of revision programming is the duality of **in** and **out** literals. The duality theorem (Theorem 3.8 from [MT98]) demonstrated that every revision program P has a counterpart, a dual revision program P^D such that P-justified revisions of a database $\mathcal{I}$ are precisely the complements of the P^D-justified revisions of the complement of $\mathcal{I}$.

The key result of this paper, the shifting theorem (Theorem 4), is a generalization of the duality theorem from [MT98]. It states that P-justified revisions of a database $\mathcal{I}$ can be computed by revising an arbitrarily chosen database $\mathcal{I}'$ by means of a certain "shifted" revision program P'. This program P' is obtained from P by uniformly replacing some literals in P by their duals. The choice of literals to replace depends on $\mathcal{I}$ and $\mathcal{I}'$. In addition, $\mathcal{I}$ and $\mathcal{I}'$ determine also a method to reconstruct P-justified revisions of $\mathcal{I}$ from P'-justified revisions of $\mathcal{I}'$.

As a special case, the shifting theorem tells us that justified revisions of arbitrary databases are determined by revisions, via shifted programs, of the empty database. This result implies two quite surprising facts. First, it means that although a revision problem is defined as pair $(P, \mathcal{I})$ (revision program and a database), full information about any revision problem can be recovered from revision problems of very special type that deal with the empty database. Moreover, the reduction does not involve any growth in the size of the revision program. Second, the shifting theorem implies the existence of a natural equivalence relation between the revision problems: two revision problems are equivalent if one can be shifted onto another.

The first of these two observations (the possibility to project revision problems onto problems with the empty database) allows us to establish a direct correspondence between revision programming and a version of logic programming proposed by Lifschitz and Woo [LW92]. We will refer to this latter system as *general disjunctive logic programming* or, simply, *general logic programming.*

In general logic programming both disjunction and negation as failure operators are allowed in the heads of rules. In this paper we study the relationship between revision programming and general logic programming. First, in Section 3, we show that revision programming is equivalent to logic programming with stable model semantics extended by a concept of a constraint. Second, in Section 4, we extend revision programming to the disjunctive case.

2 Preliminaries

In this section we will review main concepts and results concerning revision programming that are relevant to the present paper. The reader is referred to [MT98] for more details.

Elements of some finite universe U are called *atoms*. Subsets of U are called *databases*. Expressions of the form $\mathbf{in}(a)$ or $\mathbf{out}(a)$, where a is an atom, are called *literals*. Literals will be denoted by greek letters α, etc. 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 atoms $R \subseteq U$, we define

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

A set of literals is *coherent* if it does not contain a pair of dual literals. Given a database $\mathcal{I}$ and a coherent set of literals L, we define

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

Let P be a revision program. The *necessary change* of P, $NC(P)$, is the least model of P, when P is treated as a Horn program built of independent propositional atoms of the form $\mathbf{in}(a)$ and $\mathbf{out}(b)$. The necessary change describes all insertions and deletions that are enforced by the program, independently of the initial database.

In the transition from a database $\mathcal{I}$ to a database $\mathcal{R}$, the status of some elements does not change. A basic principle of revision programming is the rule of *inertia* according to which, when specifying change by means of rules in a revision program, no explicit justification for *not* changing the status is required. Explicit justifications are needed only when an atom must be inserted or deleted. The collection of all literals describing the elements that do not change the status in the transition from a database $\mathcal{I}$ to a database $\mathcal{R}$ is called the *inertia set* for $\mathcal{I}$ and $\mathcal{R}$, and is defined as follows:

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

By the *reduct* of P with respect to a pair of databases $(\mathcal{I}, \mathcal{R})$, 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 $I(\mathcal{I}, \mathcal{R})$.

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 (*a posteriori*)

justified by P in the context of the initial database $\mathcal{I}$ and a (putative) revised database $\mathcal{R}$. The database $\mathcal{R}$ is a *P-justified revision* of $\mathcal{I}$ if the necessary change of $P_{\mathcal{I},\mathcal{R}}$ is coherent and if $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$.

The following example illustrates the notion of justified revision.

Example 1. Assume that we need to form a committee. There are four people which can be on the committee: Ann, Bob, Tom and David. There are four conditions on the committee members which need to be satisfied.

First, Ann and Bob are experienced employees, and we want to see at least one of them on the committee. That is, if Ann is not on the committee, Bob must be there, and if Bob is not on the committee, Ann must be there. Second, Tom is an expert from another country and does not speak English well enough yet. So, if Tom is on the committee, David should be on the committee too, because David can serve as an interpreter. If David is not on the committee, Tom should not be there, too. Third, David asked not to be on the same committee with Ann. Fourth, Bob asked not to be on the same committee with David.

The initial proposal is to have Ann and Tom in the committee.

We want to form a committee which satisfies the four conditions and differs minimally from the initial proposal. This is a problem of computing justified revisions of initial database $\mathcal{I} = \{Ann, Tom\}$ with respect to revision program P:

$$\begin{aligned}
\mathbf{in}(Bob) &\leftarrow \mathbf{out}(Ann)\\
\mathbf{in}(Ann) &\leftarrow \mathbf{out}(Bob)\\
\mathbf{in}(David) &\leftarrow \mathbf{in}(Tom)\\
\mathbf{out}(Tom) &\leftarrow \mathbf{out}(David)\\
\mathbf{out}(Ann) &\leftarrow \mathbf{in}(David)\\
\mathbf{out}(David) &\leftarrow \mathbf{in}(Bob)
\end{aligned}$$

Let us show that $\mathcal{R} = \{Ann\}$ is a P-justified revision of $\mathcal{I}$. Clearly, $U = \{Ann, Bob, Tom, David\}$. Thus,

$$I(\mathcal{I}, \mathcal{R}) = \{\mathbf{in}(Ann), \mathbf{out}(Bob), \mathbf{out}(David)\}.$$

Therefore, $P_{\mathcal{I},\mathcal{R}}$ is the following.

$$\begin{aligned}
\mathbf{in}(Bob) &\leftarrow \mathbf{out}(Ann)\\
\mathbf{in}(Ann) &\leftarrow\\
\mathbf{in}(David) &\leftarrow \mathbf{in}(Tom)\\
\mathbf{out}(Tom) &\leftarrow\\
\mathbf{out}(Ann) &\leftarrow \mathbf{in}(David)\\
\mathbf{out}(David) &\leftarrow \mathbf{in}(Bob)
\end{aligned}$$

Hence, $NC(P_{\mathcal{I},\mathcal{R}}) = \{\mathbf{in}(Ann), \mathbf{out}(Tom)\}$. It is coherent and $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$. Consequently, $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ (in fact, unique). □

In the paper we will use the following characterizations of justified revisions given in [MT98].

Theorem 1. ([MT98]) *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. $NC(P_{\mathcal{I},\mathcal{R}}) \cup I(\mathcal{I}, \mathcal{R}) = \mathcal{R}^c$.

Two results from [MT98] are especially pertinent to the results of this paper. Given a revision program P, let us define the *dual* of P (P^D in symbols) to be the revision program obtained from P by simultaneously replacing all occurrences of all literals by their duals. The first of the two results we will quote here, the duality theorem, states that revision programs P and P^D are, in a sense, equivalent. Our main result of this paper (Theorem 4) is a generalization of the duality theorem.

Theorem 2. (Duality Theorem [MT98]) *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 $U \setminus \mathcal{R}$ is a P^D-justified revision of $U \setminus \mathcal{I}$.*

The second result demonstrates that there is a straightforward relationship between revision programs consisting of in-rules only and logic programs. Given a logic program clause c

$$p \leftarrow q_1, \ldots, q_m, not\ s_1, \ldots, not\ s_n$$

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

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

For a logic program P, we define the corresponding revision program $rp(P)$ by: $rp(P) = \{rp(c) : c \in P\}$.

Theorem 3. ([MT98]) *A set of atoms M is a stable model of a logic program P if and only if M is an $rp(P)$-justified revision of $\emptyset$.*

It is also possible to represent revision programming in logic programming. This observation is implied by complexity considerations (both the existence of a justified revision and the existence of a stable model problems are NP-complete). An explicit representation was discovered in [PT97]. In addition to representing revision rules as logic program clauses, it encodes the initial database by means of new variables and encodes the inertia rule as logic program clauses. As a consequence to our main result (Theorem 4), we obtain an alternative (and in some respects, simpler) connection between revision programming and logic programming. Namely, we establish a direct correspondence between revision programs and general logic programs of [LW92].

3 Shifting initial databases and programs

In this section we will introduce a transformation of revision programs and databases that preserves justified revisions. Our results can be viewed as a generalization of the results from [MT98] on the duality between **in** and **out** in revision programming.

Let W be a subset of U. We define a *W-transformation* on the set of all literals as follows (below, $\alpha = \mathbf{in}(a)$ or $\alpha = \mathbf{out}(a)$):

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

Thus, T_W replaces some literals by their duals and leaves other literals unchanged. Specifically, if a belongs to W then literals $\mathbf{in}(a)$ and $\mathbf{out}(a)$ are replaced by their duals.

The definition of T_W naturally extends to sets of literals and sets of atoms. Namely, for a set L of literals, we define $T_W(L) = \{T_W(\alpha)\colon \alpha \in L\}$. Similarly, for a set A of atoms, we define

$$T_W(A) = \{a : \mathbf{in}(a) \in T_W(A^c)\}.$$

The operator T_W has several useful properties. In particular, for a suitable set W, T_W allows us to transform any database $\mathcal{I}_1$ into another database $\mathcal{I}_2$. Specifically, we have:

$$T_{\mathcal{I}_1 \div \mathcal{I}_2}(\mathcal{I}_1) = \mathcal{I}_2,$$

where $\div$ denotes the symmetric difference operator. Thus, it also follows that

$$T_{\mathcal{I}}(\mathcal{I}) = \emptyset \quad \text{and} \quad T_U(\mathcal{I}) = U \setminus \mathcal{I}.$$

Some other properties of the operator T_W are gathered in the following lemma.

Lemma 1. *Let S_1 and S_2 be sets of literals. Then:*
1. $T_W(S_1 \cup S_2) = T_W(S_1) \cup T_W(S_2)$;
2. $T_W(S_1 \cap S_2) = T_W(S_1) \cap T_W(S_2)$;
3. $T_W(S_1 \setminus S_2) = T_W(S_1) \setminus T_W(S_2)$;
4. $T_W(S_1) = T_W(S_2)$ if and only if $S_1 = S_2$;
5. $T_W(T_W(S_1)) = S_1$.

In fact, Lemma 1 holds when S_1 and S_2 are sets of atoms as well.

The operator T_W can now be extended to revision rules and programs. For a revision rule $r = \alpha \leftarrow \alpha_1, \ldots, \alpha_m$, we define

$$T_W(r) = T_W(\alpha) \leftarrow T_W(\alpha_1), \ldots, T_W(\alpha_m).$$

Finally, for a revision program P, we define $T_W(P) = \{T_W(r)\colon r \in P\}$.

The main result of our paper, the shifting theorem, states that revision programs P and $T_W(P)$ are equivalent in the sense that they define essentially the same notion of change.

Theorem 4 (Shifting theorem). *Let P be a revision program. For every two databases $\mathcal{I}_1$ and $\mathcal{I}_2$, a database $\mathcal{R}_1$ is a P-justified revision of $\mathcal{I}_1$ if and only if $T_{\mathcal{I}_1 \div \mathcal{I}_2}(\mathcal{R}_1)$ is a $T_{\mathcal{I}_1 \div \mathcal{I}_2}(P)$-justified revision of $\mathcal{I}_2$.*

Proof. Let $W = \mathcal{I}_1 \div \mathcal{I}_2$. When calculating the necessary change, we treat literals as propositional atoms of the form $\mathbf{in}(a)$ and $\mathbf{out}(b)$. Observe that W-transformation can be viewed as renaming these atoms. If we rename all atoms in the Horn program, find the least model of the obtained program, and then rename the atoms back, we will get the least model of the original program.

In other words,

$$T_W(NC(P_{\mathcal{I}_1,\mathcal{R}_1})) = NC(T_W(P_{\mathcal{I}_1,\mathcal{R}_1})).$$

Let $\mathcal{R}_2 = T_W(\mathcal{R}_1)$. Observe that by the definition of T_W, $I(\mathcal{I}_2, \mathcal{R}_2) = T_W(I(\mathcal{I}_1, \mathcal{R}_1))$. Hence, $T_W(P_{\mathcal{I}_1,\mathcal{R}_1}) = (T_W(P))_{\mathcal{I}_2,\mathcal{R}_2}$.

Theorem 1 and Lemma 1 imply the following sequence of equivalences.

- $\mathcal{R}_1$ is a P-justified revision of $\mathcal{I}_1$,
- $NC(P_{\mathcal{I}_1,\mathcal{R}_1}) \cup I(\mathcal{I}_1, \mathcal{R}_1) = \mathcal{R}_1^c$,
- $T_W(\ NC(P_{\mathcal{I}_1,\mathcal{R}_1}) \cup I(\mathcal{I}_1, \mathcal{R}_1)\ ) = T_W(\mathcal{R}_1^c)$,
- $T_W(NC(P_{\mathcal{I}_1,\mathcal{R}_1}))\ \cup\ T_W(I(\mathcal{I}_1, \mathcal{R}_1)) = T_W(\mathcal{R}_1^c)$,
- $NC(T_W(P_{\mathcal{I}_1,\mathcal{R}_1}))\ \cup\ I(\mathcal{I}_2, \mathcal{R}_2) = T_W(\{\mathbf{in}(a) : a \in \mathcal{R}_1\} \cup \{\mathbf{out}(a) : a \notin \mathcal{R}_1\})$,
- $NC((T_W(P))_{\mathcal{I}_2,\mathcal{R}_2}) \cup I(\mathcal{I}_2, \mathcal{R}_2) = \mathcal{R}_2^c$,
- $\mathcal{R}_2 = T_W(\mathcal{R}_1)$ is a $T_W(P)$-justified revision of $\mathcal{I}_2$. □

Theorem 2 (the duality theorem) is a special case of Theorem 4 when $\mathcal{I}_2 = U \setminus \mathcal{I}_1$.

At first glance, a revision problem seems to have two independent parameters: a revision program P that specifies constraints to satisfy, and an initial database $\mathcal{I}$ that needs to be revised by P. The shifting theorem shows that there is a natural equivalence relation between pairs $(P, \mathcal{I})$ specifying the revision problem. Namely, a revision problem $(P, \mathcal{I})$ is *equivalent* to a revision problem $(P', \mathcal{I}')$ if $P' = T_{\mathcal{I} \div \mathcal{I}'}(P)$. This is clearly an equivalence relation. Moreover, by the shifting theorem, it follows that if $(P, \mathcal{I})$ and $(P', \mathcal{I}')$ are equivalent then P-justified revisions of $\mathcal{I}$ are in one-to-one correspondence with P'-revisions of $\mathcal{I}'$. In particular, every revision problem $(P, \mathcal{I})$ can be "projected" onto an isomorphic revision problem $(T_{\mathcal{I}}(P), \emptyset)$. Thus, the domain of all revision problems can be fully described by the revision problems that involve the empty database. There is an important point to make here. When shifting a revision program, its size does not change (in other words, all revision programs associated with equivalent revision problems have the same size).

Example 2. Let us take the same problem about forming a committee which we considered in Example 1. Recall that $\mathcal{I} = \{Ann, Tom\}$. Let us apply transformation $T_{\mathcal{I}}$ (shift to the empty initial database). It is easy to see that $T_{\mathcal{I}}(P)$ consists

of the rules:

$$\begin{aligned}
\mathbf{in}(Bob) &\leftarrow \mathbf{in}(Ann)\\
\mathbf{out}(Ann) &\leftarrow \mathbf{out}(Bob)\\
\mathbf{in}(David) &\leftarrow \mathbf{out}(Tom)\\
\mathbf{in}(Tom) &\leftarrow \mathbf{out}(David)\\
\mathbf{in}(Ann) &\leftarrow \mathbf{in}(David)\\
\mathbf{out}(David) &\leftarrow \mathbf{in}(Bob)
\end{aligned}$$

This revision program has only one justified revision of $\emptyset$, $\{Tom\}$. Observe moreover that $\{Tom\} = T_{\mathcal{J}}(\{Ann\})$. This agrees with the assertion of Theorem 4. □

There is a striking similarity between the syntax of revision programs and nondisjunctive (unitary) general logic programs of Lifschitz and Woo [LW92]. The shifting theorem, which allows us to effectively eliminate an initial database from the revision problem, suggests that both formalisms may be intimately connected. In the next section we establish this relationship. This, in turn, allows us to extend the formalism of revision programming by allowing disjunctions in the heads.

4 General disjunctive logic programs and revision programming

Lifschitz and Woo [LW92] introduced a formalism called general logic programming (see also [Lif96] and [SI95]). General logic programming deals with clauses whose heads are disjunctions of atoms (we will restrict here to the case of atoms only, even though in the original paper more general syntax is studied) and atoms within the scope of the negation-as-failure operator. Specifically, Lifschitz and Woo consider *general program rules* of the form:

$$A_1|\ldots|A_k|not\ A_{k+1}|\ldots|not\ A_l \ \leftarrow\ A_{l+1},\ldots,A_m, not\ A_{m+1},\ldots,not\ A_n, \quad (3)$$

which can be also represented as

$$HPos \cup not(HNeg) \ \leftarrow\ BPos \cup not(BNeg),$$

where $A_1,\ldots,A_n$ are atoms, $HPos = \{A_1,\ldots,A_k\}$, $HNeg = \{A_{k+1},\ldots,A_l\}$, $BPos = \{A_{l+1},\ldots,A_m\}$, $BNeg = \{A_{m+1},\ldots,A_n\}$.

A *general logic program* is defined as a collection of general program rules.

Given a set of atoms M and a clause c of the form (3), M *satisfies* c if from the fact that every A_i, $l+1 \leq i \leq m$, belongs to M and no A_i, $m+1 \leq i \leq n$, belongs to M, it follows that one of A_i, $1 \leq i \leq k$, belongs to M or one of A_i, $k+1 \leq i \leq l$, does not belong to M.

Lifschitz and Woo introduced a semantics of general logic programs that is stronger than the semantics described above. It is the semantics of *answer sets*.

Answer sets are constructed in stages. First, one defines answer sets for programs that do not involve negation as failure, that is, consist of clauses

$$A_1 | \ldots | A_k \ \leftarrow \ A_{k+1}, \ldots, A_m \tag{4}$$

Given a program P consisting of clauses of type (4), a set of atoms M is an answer set for P if M is a minimal set of atoms satisfying all clauses in P.

Next, given a general logic program P (now possibly with negation as failure operator) and a set of atoms M, one defines the *reduct* of P with respect to M, denoted P^M, as the general logic program without negation as failure obtained from P by

- deleting each disjunctive rule such that $HNeg \not\subseteq M$ or $BNeg \cap M \neq \emptyset$, and
- replacing each remaining disjunctive rule by $HPos \ \leftarrow \ BPos$.

A set of atoms M is an *answer set* for P if M is an answer set for P^M.

4.1 Answer sets for general programs and justified revisions

We will now show that revision programming is closely connected with a special class of general logic programs, namely those for which all rules have a single atom in the head. We will call such rules and programs *unitary*.

The encoding of revision rules as general logic program clauses is straightforward. Given a revision program in-rule r:

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

we define the disjunctive rule $dj(r)$ as:

$$p \leftarrow q_1, \ldots, q_m, not\ s_1, \ldots, not\ s_n.$$

Similarly, given a revision program out-rule r:

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

we define the disjunctive rule $dj(r)$ as:

$$not\ p \leftarrow q_1, \ldots, q_m, not\ s_1, \ldots, not\ s_n.$$

Finally, for a revision program P, define $dj(P) = \{dj(r) \colon r \in P\}$.

The mapping $dj(\cdot)$ is a 1-1 correspondence between revision rules and unitary general logic program rules, and revision programs and unitary general logic programs.

The following result states that revision problems where the initial database is empty can be dealt with by means of general logic programs. This result can be viewed as a generalization of Theorem 3.

Theorem 5. *Let P be a revision program. Then, $\mathcal{R}$ is a P-justified revision of $\emptyset$ if and only if $\mathcal{R}$ is an answer set for $dj(P)$.*

Proof. Let $\mathcal{R}$ be a database. Let P' be a revision program obtained from P by deleting each out-rule that has $\mathbf{out}(a)$ in the head for some $a \notin \mathcal{R}$, and deleting each rule which has $\mathbf{out}(a)$ in the body for some $a \in \mathcal{R}$. Then, $dj(P')$ is the disjunctive program obtained from $dj(P)$ by deleting each disjunctive rule such that $HNeg \not\subseteq \mathcal{R}$ or $BNeg \cap \mathcal{R} \neq \emptyset$ (recall that this is the first step in constructing $dj(P)^{\mathcal{R}}$).

Observe that $\mathcal{R}$ is P-justified revision of $\emptyset$ if and only if $\mathcal{R}$ is P'-justified revision of $\emptyset$. Indeed, inertia $I(\emptyset, \mathcal{R}) = \{\mathbf{out}(a) : a \notin \mathcal{R}\}$. From Theorem 1 we have that $\mathcal{R}$ is P-justified revision of $\emptyset$ if and only if

$$NC(P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}) \;=\; NC(P \cup \{\mathbf{out}(a) \leftarrow : a \notin \mathcal{R}\}) \;=\; \mathcal{R}^c.$$

From the definition of P' and the fact that $NC(P \cup \{\mathbf{out}(a) \leftarrow : a \notin \mathcal{R}\})$ is coherent (as it equals $\mathcal{R}^c$), it follows that

$$NC(P \cup \{\mathbf{out}(a) \leftarrow : a \notin \mathcal{R}\}) \;=\; NC(P' \cup \{\mathbf{out}(a) \leftarrow : a \notin \mathcal{R}\}).$$

Therefore, using Theorem 1 again, we get that $\mathcal{R}$ is P-justified revision of $\emptyset$ if and only if $\mathcal{R}$ is P'-justified revision of $\emptyset$.

Observe that if literal $\mathbf{out}(a)$, for some a, occurs in the body of a rule in P' then $\mathbf{out}(a) \in I(\emptyset, \mathcal{R})$. Also, inertia $I(\emptyset, \mathcal{R})$ consists only of literals of the form $\mathbf{out}(a)$. Therefore, $P'_{\emptyset,\mathcal{R}}$ is obtained from P' by eliminating each literal of the form $\mathbf{out}(a)$ from the bodies of the rules.

Let $P'_{\emptyset,\mathcal{R}} = P'' \cup P'''$, where P'' consists of all in-rules of $P'_{\emptyset,\mathcal{R}}$, $P''' = P'_{\emptyset,\mathcal{R}} \setminus P''$ consists of all out-rules of $P'_{\emptyset,\mathcal{R}}$. Note, that all rules from P'' and P''' have only literals of the form $\mathbf{in}(a)$ in their bodies. Observe that if $r \in P'''$, then its head, $head(r) = \mathbf{out}(a)$ for some $a \in \mathcal{R}$. By the definition, $dj(P)^{\mathcal{R}}$ is obtained from $dj(P')$ by replacing each disjunctive rule by $HPos \leftarrow BPos$. Therefore,

$$dj(P)^{\mathcal{R}} \;=\; dj(P'') \cup \{ \leftarrow a_1, \ldots, a_k : \mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_k) \in P'''\}.$$

After this observations we are ready to prove the statement of the theorem.
($\Rightarrow$) Let $\mathcal{R}$ be a P-justified revision of $\emptyset$. It follows that $\mathcal{R}$ is a P'-justified revision of $\emptyset$. Thus, $\mathcal{R} = \emptyset \oplus NC(P'_{\emptyset,\mathcal{R}})$. Assume that there exists a literal $\mathbf{out}(a) \in NC(P'_{\emptyset,\mathcal{R}})$. Since $NC(P'_{\emptyset,\mathcal{R}})$ is a subset of heads of rules from $P'_{\emptyset,\mathcal{R}}$, it must be the case that $a \in \mathcal{R}$. This contradicts the fact that the necessary change is coherent and $\mathcal{R} = \emptyset \oplus NC(P'_{\emptyset,\mathcal{R}})$. Therefore, $NC(P'_{\emptyset,\mathcal{R}})$ consists only of literals of the form $\mathbf{in}(a)$. It implies that $NC(P'_{\emptyset,\mathcal{R}}) = \{\mathbf{in}(a) : a \in \mathcal{R}\}$ is the least model of P'', and for every rule $r \in P'''$ there exist b such that $\mathbf{in}(b) \in body(r)$ and $b \notin \mathcal{R}$. Hence, $\mathcal{R}$ is the minimal set of atoms which satisfies all clauses in $dj(P)^{\mathcal{R}}$. Thus, $\mathcal{R}$ is an answer set for $dj(P)$.
($\Leftarrow$) Let $\mathcal{R}$ be an answer set for $dj(P)$. That is, $\mathcal{R}$ is the minimal set of atoms which satisfies all clauses in

$$dj(P'') \cup \{ \leftarrow a_1, \ldots, a_k : \mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_k) \in P'''\}.$$

Then, any subset of $\mathcal{R}$ satisfies all clauses in

$$\{ \leftarrow a_1, \ldots, a_k : \mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_k) \in P'''\}.$$

Therefore, $\mathcal{R}$ is the minimal set of atoms which satisfies all clauses in $dj(P'')$. Hence, $\{\mathbf{in}(a) : a \in \mathcal{R}\}$ is the least model of P'' and satisfies P'''. Consequently, $\{\mathbf{in}(a) : a \in \mathcal{R}\} = NC(P'_{\emptyset,\mathcal{R}})$, and $\mathcal{R} = \emptyset \oplus NC(P'_{\emptyset,\mathcal{R}})$. By the definition, $\mathcal{R}$ is P'-justified revision of $\emptyset$. Therefore, $\mathcal{R}$ is P-justified revision of $\emptyset$. □

It might appear that the scope of Theorem 5 is restricted to the special case of revision programs that update the empty database. However, the shifting theorem allows us to extend this result to the general case. Thus, revision programming turns out to be equivalent to the unitary fragment of general logic programming. Indeed, we have the following corollary.

Corollary 1. *Let P be a revision program and $\mathcal{I}$ a database. Then, a database $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $T_{\mathcal{I}}(\mathcal{R})$ is an answer set for the program $dj(T_{\mathcal{I}}(P))$.*

Consider a revision program P and a database $\mathcal{I}$. A rule $r \in P$ is called a *constraint* (with respect to $\mathcal{I}$) if its head is of the form $\mathbf{in}(a)$, for some $a \in \mathcal{I}$, or $\mathbf{out}(a)$, for some $a \notin \mathcal{I}$.

Theorem 6. *Let P be a revision program and let $\mathcal{I}$ be a database. Let P' consist of all rules in P that are constraints with respect to $\mathcal{I}$. Let $P'' = P \backslash P'$. A database $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $\mathcal{R}$ is a P''-justified revision of $\mathcal{I}$ that satisfies all rules from P'.*

Proof. By the shifting theorem it is enough to prove the statement for the case $\mathcal{I} = \emptyset$. Let $\mathcal{I} = \emptyset$. Then, P' consists of all out-rules of P and P'' consists of all in-rules of P.
($\Rightarrow$) If $\mathcal{R}$ is a P-justified revision of $\emptyset$, then $\mathcal{R}$ is a model of P. Hence, it is a model of $P' \subseteq P$.

Theorem 1 implies that

$$NC(P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}) = \{\mathbf{in}(a) : a \in \mathcal{R}\} \cup \{\mathbf{out}(a) : a \notin \mathcal{R}\}.$$

Let $M = NC(P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\})$. That is, M is the least model of

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

By the definition of inertia, $I(\emptyset, \mathcal{R}) = \{\mathbf{out}(a) : a \notin \mathcal{R}\}$.

We will now show that M is the least model of $P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}$.

Let us divide P' into two disjoint parts: $P' = P'_1 \cup P'_2$, where heads of the rules from P'_1 are in $\{\mathbf{out}(a) : a \in \mathcal{R}\}$ and heads of the rules from P'_2 are in $\{\mathbf{out}(a) : a \notin \mathcal{R}\}$. For each rule $r \in P'_2$, $head(r) \in I(\emptyset, \mathcal{R})$. Hence, there exists rule $head(r) \leftarrow$ in the set $\{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}$. Therefore, M is also the least model of the program

$$P'' \cup P'_1 \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}.$$

If we remove from the program some rules whose premises are false in M, M remains the least model of the reduced program. Let us show that premises

of all rules from P_1' are false in M. Indeed, let r be a rule from P_1'. Then, $head(r) \in \{\mathbf{out}(a) : a \in \Re\}$. Assume that premises of r are true in M. Then, $head(r)$ must be true in M, since M is the model of $P'' \cup P_1' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \Re)\}$. Hence, $M \cap \{\mathbf{out}(a) : a \in \Re\} \neq \emptyset$, which contradicts the fact that $M = \{\mathbf{in}(a) : a \in \Re\} \cup \{\mathbf{out}(a) : a \notin \Re\}$. Therefore, M is the least model of the program $P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \Re)\}$. In other words,

$$NC(P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \Re)\}) = \{\mathbf{in}(a) : a \in \Re\} \cup \{\mathbf{out}(a) : a \notin \Re\}.$$

From Theorem 1 we conclude that $\Re$ is a P''-justified revision of $\emptyset$.
($\Leftarrow$) Assume $\Re$ is a P''-justified revision of $\emptyset$, and $\Re$ satisfies all rules from P'. Theorem 1 implies that

$$NC(P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \Re)\}) = \Re^c.$$

Let $M = \Re^c$. Then, M is the least model of

$$P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \Re)\}.$$

Clearly, M is also the least model of a modified program obtained by adding some rules that are satisfied by M. All rules in P' are satisfied by M by our assumption. Therefore, M is the least model of

$$P' \cup P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \Re)\} = P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \Re)\}.$$

Hence,

$$NC(P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \Re)\}) = M = \Re^c.$$

By Theorem 1, $\Re$ is a P-justified revision of $\emptyset$. □

The reason for the term "constraint" is now clear. In computing P-justified revisions only "non-constraints" are used. Then, the constraint part of P is used to weed out some of the computed revisions.

Clearly, if $\mathcal{I} = \emptyset$, the constraints are exactly the out-rules of a revision program. We can extend the notion of a constraint to the case of unitary general logic programs. Namely, a unitary program rule is a *constraint* if its head is of the form *not a* (note that this notion of constraint is different from the one used in [Lif96]). Theorem 6 has the following corollary.

Corollary 2. *Let P be a unitary general logic program and let P' consists of all constraints in P. A set M is an answer set for P if and only if M is a stable model for $P \setminus P'$ that satisfies P'.*

It follows from the shifting theorem and from Theorem 5 that in order to describe updates by means of revision programming, it is enough to consider logic programs with stable model semantics and rules with *not a* in the heads that work as constraints.

Corollary 3. *Let P be a revision program and let $\mathcal{I}$ be a database. Then, a database $\Re$ is a P-justified revision of $\mathcal{I}$ if and only if $T_{\mathcal{I}}(\Re)$ is a stable model of the logic program $dj(T_{\mathcal{I}}(P) \setminus P')$ that satisfies P', where P' consists of all constraints in $T_{\mathcal{I}}(P)$.*

4.2 Disjunctive revision programs

The results of Section 4.1 imply an approach to extend revision programming to include clauses with disjunctions in the heads. Any such proposal must satisfy several natural postulates. First, the semantics of disjunctive revision programming must reduce to the semantics of justified revisions on disjunctive revision programs consisting of rules with a single literal in the head. Second, the shifting theorem must generalize to the case of disjunctive revision programs. Finally, the results of Section 4.1 indicate that there is yet another desirable criterion. Namely, the semantics of disjunctive revision programming over the empty initial database must reduce to the Lifschitz and Woo semantics for general logic programs. The construction given below satisfies all these three conditions.

First, let us introduce the syntax of disjunctive revision programs. By a *disjunctive revision rule* we mean an expression of the following form:

$$\alpha_1 | \ldots | \alpha_m \leftarrow \alpha_{m+1}, \ldots, \alpha_n \ , \tag{5}$$

where $\alpha_i,\ 1 \leq i \leq n$ are literals (that is, expressions of the form $\mathbf{in}(a)$ or $\mathbf{out}(a)$). A *disjunctive revision program* is a collection of disjunctive revision rules.

In order to specify semantics of disjunctive revision programs we first define the closure of a set of literals under a disjunctive rule. A set L of literals is *closed* under a rule (5) if at least one α_i, $1 \leq i \leq m$, belongs to L or if at least one α_i, $m+1 \leq i \leq n$, does *not* belong to L. A set of literals L is *closed* under a disjunctive revision program P if it is closed under all rules of P.

The next step involves the generalization of the notion of necessary change. Let P be a disjunctive revision program. A *necessary change* entailed by P is any minimal set of literals that is closed under P. Notice that in the context of disjunctive programs the necessary change may not be unique.

Recall that a database is a collection of atoms from universe U. A literal l is *satisfied* by a database $R \subseteq U$ if $l = \mathbf{in}(a)$ and $a \in \mathcal{R}$, or $l = \mathbf{out}(a)$ and $a \notin \mathcal{R}$, for some $a \in U$. We say that the body of a disjunctive revision rule is *satisfied* by a database R if every literal from the body is satisfied by R.

We will now introduce the notion of a reduct of a disjunctive revision program P with respect to two databases $\mathcal{I}$ (initial database) and $\mathcal{R}$ (a putative revision of $\mathcal{I}$). The reduct, denoted by $P^{\mathcal{I},\mathcal{R}}$, is constructed in the following four steps.

Step 1: Eliminate from the body of each rule in P all literals in $I(\mathcal{I},\mathcal{R})$.
Step 2: Remove all rules r, such that $head(r) \cap I(\mathcal{I},\mathcal{R}) \neq \emptyset$.
Step 3: Eliminate from the remaining rules every rule whose body is not satisfied by $\mathcal{R}$.
Step 4: Remove from the heads of the rules all literals that are not satisfied by $\mathcal{R}$.

We are ready now to define the notion a P-justified revision of a database $\mathcal{I}$ for the case of disjunctive revision programs. Let P be a disjunctive revision program. A database $\mathcal{R}$ is a *P-justified revision* of a database $\mathcal{I}$ if for some coherent necessary change L of $P^{\mathcal{I},\mathcal{R}}$, $\mathcal{R} = \mathcal{I} \oplus L$. Let us observe that only steps

(1) and (2) in the definition of reduct are important. Steps (3) and (4) do not change the defined notion of revision but lead to a simpler program.

The next example illustrates a possible use of disjunctive revision programming.

Example 3. Let us now represent the situation of Example 1 as a disjunctive revision program P:

$$\begin{aligned} \mathbf{in}(Ann) \mid \mathbf{in}(Bob) &\leftarrow \\ \mathbf{out}(Tom) \mid \mathbf{in}(David) &\leftarrow \\ \mathbf{out}(Ann) &\leftarrow \mathbf{in}(David) \\ \mathbf{out}(David) &\leftarrow \mathbf{in}(Bob) \end{aligned}$$

Assume that $\mathcal{I} = \{Ann, Tom\}$, $\mathcal{R} = \{Ann\}$. Then, inertia $I(\mathcal{I}, \mathcal{R}) = \{\mathbf{in}(Ann), \mathbf{out}(Bob), \mathbf{out}(David)\}$. The reduct $P^{\mathcal{I},\mathcal{R}} = \{\mathbf{out}(Tom) \leftarrow\}$. The only necessary change of $P^{\mathcal{I},\mathcal{R}}$ is $L = \{\mathbf{out}(Tom)\}$. Since L is coherent and $\mathcal{R} = \mathcal{I} \oplus L$, $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$. □

The following three theorems show that the semantics for disjunctive revision programs described here satisfies the three criteria described above.

Theorem 7. *Let P be a revision program (without disjunctions). Then, $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ when P is treated as a disjunctive revision program.*

Proof. For any revision program P (without disjunctions), the least model of P, when treated as a Horn program built of independent propositional atoms of the form $\mathbf{in}(a)$ and $\mathbf{out}(a)$, is closed under P. Moreover, every set of literals that is closed under P must contain the least model of P. Therefore, the notions of necessary change coincide for revision programs without disjunctions, when treated as ordinary revision programs and as disjunctive revision programs. Hence, the notions of justified revisions coincide, too. □

The definition of T_W naturally extends to the case of disjunctive revision programs.

Theorem 8 (Shifting theorem). *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be databases, and let P be a disjunctive revision program. Let $W = \mathcal{I}_1 \div \mathcal{I}_2$. Then, $\mathcal{R}_1$ is P-justified revision of $\mathcal{I}_1$ if and only if $T_W(\mathcal{R}_1)$ is $T_W(P)$-justified revision of $\mathcal{I}_2$.*

Proof. Similarly to the case of ordinary revision programs, in computing justified revisions for disjunctive revision programs we are dealing with literals. W-transformation can be viewed as renaming these literals, which does not effect the procedure. Therefore, the statement of the theorem holds. □

The embedding of (unitary) revision programs extends to the case of disjunctive revision programs. As before, each literal $\mathbf{in}(a)$ is replaced by the corresponding atom a and each literal $\mathbf{out}(a)$ is replaced by *not* a. The general logic

program obtained in this way from a disjunctive revision program P is denoted by $dj(P)$.

Theorem 9. *Let P be a disjunctive revision program. Then, $\mathcal{R}$ is a P-justified revision of $\emptyset$ if and only if $\mathcal{R}$ is an answer set for $dj(P)$.*

Proof. First notice that for every $\mathcal{R}$, $I(\emptyset, \mathcal{R})$ is equal to $\{\mathbf{out}(a) : a \notin \mathcal{R}\}$.

Observe that step 2 in the definition of the reduct $P^{\mathcal{I},\mathcal{R}}$ removes exactly those rules r for which $dj(r)$ satisfies condition $HNeg \not\subseteq \mathcal{R}$.

Step 3 removes all rules r for which $dj(r)$ satisfies condition $BNeg \cap \mathcal{R} \neq \emptyset$, as well as rules containing $\mathbf{in}(a)$ in the bodies for some $a \notin \mathcal{R}$ (corresponding disjunctive logic program rules have a in the bodies for some $a \notin \mathcal{R}$).

Step 1 eliminates from the bodies of the rules of P all literals that are in $I(\mathcal{I}, \mathcal{R})$. In disjunctive logic program it corresponds to eliminating $not(BNeg)$ parts from the bodies of the remaining rules.

Step 4 in particular corresponds to eliminating $not(HNeg)$ parts from the heads of the remaining disjunctive logic program rules.

Therefore, $dj(P)^{\mathcal{R}}$, when compared to $dj(P^{\mathcal{I},\mathcal{R}})$, may only have some extra rules, the bodies of which are not satisfied by $\mathcal{R}$, or some extra literals in the heads, which are not satisfied by $\mathcal{R}$. Hence, the statement of the theorem holds. □

We conclude this section with a simple observation related to the computational complexity of a problem of existence of justified revisions in the case of disjunctive revision programming. We will show that disjunctive revision programming is an essential extension of the unitary revision programming. In [MT98] it was proved that the problem of existence of a justified revision in the case of unitary revision programming is NP-complete. Using the results of Eiter and Gottlob [EG95] and our correspondence between disjunctive revision programs and general logic programs we obtain the following result.

Theorem 10. *The following problem is Σ_2^P-complete: Given a finite disjunctive revision program and a database $\mathcal{I}$, decide whether $\mathcal{I}$ has a P-justified revision.*

It follows that disjunctive revision programming is an essential extension of the unitary revision programming (unless the polynomial hierarchy collapses).

5 Future work

Lifschitz, Tang and Turner [LTT97] extended the answer set semantics to a class of logic programs with nested expressions permitted in the bodies and heads of rules. It can be shown that our formalism can be lifted to revision programs admitting nested occurrences of connectives as well.

The connections between revision programming and logic programming, presented in this work, imply a straightforward approach to compute justified revisions. Namely, a revision problem $(P, \mathcal{I})$ must first be compiled into a general logic program (by applying the transformation $T_{\mathcal{I}}$ to P). Then, answer sets to $T_{\mathcal{I}}(P)$ must be computed and "shifted" back by means of $T_{\mathcal{I}}$.

To compute the answer sets of the general logic program $T_{\mathrm{J}}(P)$, one might use any of the existing systems computing stable models of logic programs (for instance s-models [NS96], DeReS [CMMT95], and for disjunctive case DisLoP [ADN97], or a system `dlv` presented in [ELM+97]). Some care needs to be taken to model rules with negation as failure operator in the heads as standard logic program clauses or defaults.

In our future work, we will investigate the efficiency of this approach to compute justified revisions and we will develop related techniques tailored specifically for the case of revision programming.

References

[ADN97] C. Aravindan, J. Dix, and I. Niemelä. DisLoP: Towards a disjunctive logic programming system. In *Logic programming and nonmonotonic reasoning (Dagstuhl, Germany, 1997)*, volume 1265 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 1997.

[ALP+98] J.J. Alferes, J.A. Leite, L.M. Pereira, H. Przymusinska, and T.C. Przymusinski. Dynamic logic programming. Accepted at KR'98: Sixth International Conference on Principles of Knowledge Representation and Reasoning, Trento, Italy, June 1998.

[AP97] J.J. Alferes and L.M. Pereira. Update-programs can update programs. In *Non-Monotonic Extensions of Logic Programming (Bad Honnef, 1996)*, volume 1216 of *Lecture Notes in Computer Science*, pages 110–131, Berlin, 1997. Springer.

[Bar97] C. Baral. Embedding revision programs in logic programming situation calculus. *Journal of Logic Programming*, 30(1):83–97, 1997.

[BM97] N. Bidoit and S. Maabout. Update programs versus revision programs. In *Non-monotonic extensions of logic programming (Bad Honnef, 1996)*, volume 1216 of *Lecture Notes in Computer Science*, pages 151–170, Berlin, 1997. Springer.

[CMMT95] P. Cholewiński, W. Marek, A. Mikitiuk, and M. Truszczyński. Experimenting with nonmonotonic reasoning. In *Logic programming (Kanagawa, 1995)*, MIT Press Series in Logic Programming, pages 267–281, Cambridge, MA, 1995. MIT Press.

[EG95] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[ELM+97] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for non-monotonic reasoning. In *Logic programming and nonmonotonic reasoning (Dagstuhl, Germany, 1997)*, volume 1265 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 1997.

[Lif96] V. Lifschitz. Foundations of logic programming. In *Principles of Knowledge Representation*, pages 69–127. CSLI Publications, 1996.

[LTT97] V. Lifschitz, L. R. Tang, and H. Turner. Nested expressions in logic programs. unpublished draft, 1997.

[LW92] V. Lifschitz and T.Y.C. Woo. Answer sets in general nonmonotonic reasoning. In *Proceedings of the 3rd international conference on principles of knowledge representation and reasoning, KR '92*, pages 603–614, San Mateo, CA, 1992. Morgan Kaufmann.

[MT94] W. Marek and M. Truszczyński. Revision specifications by means of programs. In *Logics in artificial intelligence (York, 1994)*, volume 838 of *Lecture Notes in Computer Science*, pages 122–136, Berlin, 1994. Springer.

[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.

[McCT95] N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *IJCAI-95, Vol. 1, 2 (Montreal, PQ, 1995)*, pages 1978–1984, San Francisco, CA, 1995. Morgan Kaufmann.

[MT98] W. Marek and M. Truszczyński. Revision programming. *Theoretical Computer Science*, 190(2):241–277, 1998.

[NS96] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In *Proceedings of JICSLP-96*. MIT Press, 1996.

[PT97] T. C. Przymusinski and H. Turner. Update by means of inference rules. *Journal of Logic Programming*, 30(2):125–143, 1997.

[SI95] C. Sakama and K. Inoue. Embedding circumscriptive theories in general disjunctive programs. In *Logic programming and nonmonotonic reasoning (Lexington, KY, 1995)*, volume 928 of *Lecture Notes in Computer Science*, pages 344–357, Berlin, 1995. Springer.

[Tur97] H. Turner. Representing actions in logic programs and default theories: a situation calculus approach. *Journal of Logic Programming*, 31(1-3):245–298, 1997.