Revision programming = logic programming + 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 revisions can be shifted, that is for any program P, for any pair of initial databases I and I' we can shift the program P to program P' so that the size of the resulting program does not increase and so that P-justified revisions of I are shifted to P'-justified revisions of 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 with stable model semantics extended by the concept of a constraint. Finally, we use the connection between revision programming and general logic programming to introduce disjunctive and nested versions 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:

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

or

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

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, 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 I, and none of the data items b_j, $1 \leq j \leq n$, belongs to I. Then, to enforce the constraint (1), the item a must be added to the database (removed from it, in the case of the constraint (2)), rather then 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 I, this semantics specifies a family of databases, each of which might be chosen as an update of I by means of the program P. These revised databases are called P-justified revisions of I. In [MT98] (and in the earlier papers [MT94] and [MT95a]), basic properties of justified revisions were established. Subsequently, revision programming was studied in the context of situation calculus [Bar97] and reasoning about actions [MT95b, Tur97].

Revision programming has also been investigated from the perspective of its close relationship with logic programming. In [MT98], we argued that revision programming extends logic programming with stable semantics. Namely, we argued 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 [BM96].

We have already mentioned the key property of revision programming — the duality of **in** and **out** literals. The duality theorem 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 I are precisely the complements of the P^D-justified revisions of the complement of I.

The key result of this paper, the shifting theorem (Theorem 3.2), is a generalization of the duality theorem from [MT98]. It states that P-justified revisions of a database I can be computed by revising an arbitrarily chosen database 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. Which literals to replace depends of I and I'. In addition, I and I' determine also a method to reconstruct P-justified revisions of I from P'-justified revisions of 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 implies two quite surprising facts. First, it means that although a revision problem is defined as pair (P, 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, we show that revision programming is equivalent to logic programming with stable model semantics extended by a concept of a constraint. Second, we extend revision programming to the disjunctive case. This is presented in Section 4. Finally, we use our understanding of the relationship to generalize Lifschitz's construction of nested operators [VLT97] to the context of revision programming. We will briefly outline our approach in Section 5.

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*. Expressions of the form **in**(a) or **out**(a), where a is an atom, are called literals. For a literal **in**(a), its *dual* is the literal **out**(a). Similarly, the *dual* of **out**(a) is **in**(a). The dual of a literal α is denoted by α^D.

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

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

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 $\textbf{in}(a)$ and $\textbf{out}(b)$. 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 I to a database 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 I to a database R is called the *inertia set* for I and R, and is defined as follows:

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

By the *reduct* of P with respect to a pair of databases (I, R), denoted by $P_{I,R}$, we mean the revision program obtained from P by eliminating from the body of each rule in P all literals in $I(I, R)$.

The necessary change of the program $P_{I,R}$ provides a justification for some insertions and deletions. These are exactly the changes that are (*a posteriori*) justified by P in the context of the initial database I and a (putative) revised database R. The database R is a P-justified revision of I if the necessary change of $P_{I,R}$ is coherent and if $R = I \oplus \textit{NC}(P_{I,R})$.

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 3.2) is a generalization of the duality theorem.

Theorem 2.1 (Duality Theorem [MT98]) *Let P be a revision program and let I be a database. Then, R is a P-justified revision of I if and only if $\overline{R}$ is a P^D-justified revision of $\overline{I}$.*

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

$$p \leftarrow q_1, \ldots, q_m, \textbf{not}(s_1), \ldots, \textbf{not}(s_n)$$

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

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

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

Theorem 2.2 ([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 3.2), 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 literals as follows (below, $\alpha = \textbf{in}(a)$ or $\alpha = \textbf{out}(a)$):

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

Thus, T_W replaces some literals (those, built of elements of W) by their duals and leaves other literals unchanged.

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

$$T_W(A) = \{a\colon \textbf{in}(a) \in T_W(A^c)\},$$

where $A^c = \{\textbf{in}(a)\colon a \in A\} \cup \{\textbf{out}(a)\colon a \notin A\}$.

The operator T_W has several useful properties. In particular, for a suitable set W, T_W allows us to transform any database I_1 into another database I_2. Specifically, we have:

$$T_{I_1 \div I_2}(I_1) = I_2, \tag{3}$$

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

$$T_I(I) = \emptyset \quad \text{and} \quad T_U(I) = \bar{I}, \tag{4}$$

where $\bar{I} = U \setminus I$.

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

Lemma 3.1 *Let S_1 and S_2 be sets of literals (or sets of atoms). 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. $T_W(T_W(S_1)) = S_1$*;*

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

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

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 3.2 (Shifting theorem) *Let P be a revision program. For every two databases I_1 and I_2, a database R is a P-justified revision of I_1 if and only if $T_{I_1 \div I_2}(R)$ is a $T_{I_1 \div I_2}(P)$-justified revision of I_2.*

The duality theorem 2.1 is a special case of Theorem 3.2 for when $I_2 = U$.

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 I that needs to be revised by P. The shifting theorem shows that there is a natural equivalence relation between pairs (P, I) specifying the revision problem. Namely, a revision problem (P, I) is *equivalent* to a revision problem (P', I') if $P' = T_{I \div I'}(P)$. This is clearly an equivalence relation. Moreover, by shifting theorem, it follows that if (P, I) and (P', I') are equivalent then P-justified revisions of I are in one-to-one correspondence with P'-revisions of I'. In particular, every revision problem (P, I) can be "projected" onto an isomorphic revision problem $(T_I(P), \emptyset)$. 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 3.3 Let P be a revision program consisting of the following rules:

out$(a) \leftarrow$ **in**(b)
in$(c) \leftarrow$ **out**(b)
out$(d) \leftarrow$ **in**(c).

Consider a database $I = \{a, b\}$. The only P-justified revision of I is $R = \{b\}$. It is easy to see that $T_I(P)$ consists of the rules:

in$(a) \leftarrow$ **out**(b)
in$(c) \leftarrow$ **in**(b)
out$(d) \leftarrow$ **in**(c).

This revision program has only one justified revision of $T_I(I) = \emptyset$: $\{a\}$. Observe moreover that $\{a\} = T_I(\{b\})$, which agrees with the assertion of Theorem 3.2. $\Box$

There is a striking similarity between the syntax of revision programs and nondisjunctive (unitary) general logic programs of Lifschitz and Woo [LW92]. 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 literals (from some first-order logic language; we will restrict here to the propositional case only) and literals within the scope of the negation-as-failure operator. Specifically, Lifschitz and Woo consider general program rules of the form:

$$L_1|\ldots|L_k|not\ L_{k+1}|\ldots|not\ L_l \ \leftarrow\ L_{l+1},\ldots,L_m, not\ L_{m+1},\ldots,not\ L_n. \tag{5}$$

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

Given a set of literals M and a clause c of the form (5), M *satisfies* C if from the fact that every L_i, $l+1 \leq i \leq m$, belongs to M and no L_i, $m+1 \leq i \leq m$, belongs to M, it follows that one of L_i, $1 \leq i \leq k$, belongs to M or one of L_i, $k+1 \leq i \leq l$, does not belong to M.

Lifschitz and Woo introduce 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

$$L_1|\ldots|L_k \ \leftarrow\ L_{k+1},\ldots,L_m \tag{6}$$

Given a program P consisting of clauses of type (6), a set of literals M is an answer set for P if M is a minimal set of literals satisfying all clauses in P and contains all literals that it entails.

Next, given a general logic program P (now possibly with negation as failure operator) and a set of literals M, one defines the *reduct* of P with respect to M, denoted P^M (this reduct is a generalization of the familiar Gelfond-Lifschitz reduct, see [LW92] for details). The reduct P^M consists of clauses of type (6) only. A set of literals 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 literal in the head. We will call such rules *unitary*.

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

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

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:

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

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 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 2.2.

Theorem 4.1 *Let P be a revision program. Then R is a P-justified revision of $\emptyset$ if and only if R is an answer set for $dj(P)$.*

It might appear that the scope of Theorem 4.1 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 4.2 *Let P be a revision program and I a database. Then a database J is a P-justified revision of I if and only if $I \div J$ is an answer set for the program $dj(T_I(P))$.*

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

Theorem 4.3 *Let P be a revision program and let I be a database. Let P' consist of all rules in P that are constraints with respect to I. Let $P'' = P \setminus P'$. A database R is a P-justified revision of I if and only if R is a P''-justified revision of I that satisfies all rules from P'.*

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 $I = \emptyset$, 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 4.3 has the following corollary.

Corollary 4.4 *Let P be a unitary general logic program without classical negation 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 4.1 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 4.5 *Let P be a revision program and let I be a database. Then, a database R is a P-justified revision of I if and only if $T_I(R)$ is a stable model of the logic program $T_I(P) \setminus P'$ that satisfies P', where P' consists of all constraints in $T_I(P)$.*

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:

$$\alpha_1|\ldots|\alpha_m \leftarrow \alpha_{m+1},\ldots,\alpha_n \ , \tag{7}$$

where $\alpha_i, \ 1 \leq i \leq n$ are *revision* literals (that is, expressions of the form **in**(a) or **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 closure of a set of revision literals under a disjunctive rule. A set L of literals is *closed* under a rule (7) if at least one α_i, $1 \leq i \leq m$, belongs to M or if at least one α_i, $m+1 \leq i \leq n$, does *not* belong to M. 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 revision literals that is closed under P. Notice that in the context of disjunctive programs the necessary change may not be unique.

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

Step 1: Eliminate from the body of each rule in P all literals in $I(I,R)$.

Step 2: Remove all rules r, such that $head(r) \cap I(I,R) \neq \emptyset$.

Step 3: Eliminate from the remaining rules every rule whose body is not satisfied by R.

Step 4: Remove from the heads of the rules all literals that contradict R.

We are ready now to define the notion a P-justified revision of a database I for the case of disjunctive revision programs. Let P be a disjunctive revision program. A database R is a P-justified revision of a database I if for some coherent necessary change L of $P^{I,R}$, $R = I \oplus L$. 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 4.6 Consider a manager whose office staff consists of Ann and Chen. The office does not work well and the manager wants to do some changes. She is aware of the following constraints. She needs Ann or Bob. She must select David or fire Chen. If David is selected then Ann must be transferred. Finally, if Bob is selected, David must be transferred. These constraints can be captured by a disjunctive program P:

$$\begin{array}{rcl} in(Ann) \mid in(Bob) & \leftarrow & \\ out(Chen) \mid in(David) & \leftarrow & \\ out(Ann) & \leftarrow & in(David) \\ out(David) & \leftarrow & in(Bob) \end{array}$$

Moreover, we have $I = \{Ann, Chen\}$ (initial database). Consider $R = \{Ann\}$. Then inertia $I(I, R) = \{in(Ann), out(Bob), out(David)\}$. The reduct $P^{I,R} = \{out(Chen) \leftarrow\}$. The only necessary change of $P^{I,R}$ is $L = \{out(Chen)\}$. Since L is coherent and $R = I \oplus L$, R is a P-justified revision of I (in fact, unique). □

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

Theorem 4.7 *Let P be a revision program (without disjunctions). Then R is a P-justified revision of I if and only if R is a P-justified revision of I when P is treated as a disjunctive revision program.*

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

Theorem 4.8 (Shifting theorem) *Let I_1 and I_2 be databases, P - disjunctive revision program. Let $W = I_1 \div I_2$. Then, R_1 is P-justified revision of I_1 if and only if $T_W(R_1)$ is $T_W(P)$-justified revision of I_2.*

The embedding of (unitary) revision programs extends to the case of disjunctive revision programs. As before, each literal **in**(a) is replaced by the corresponding atom a and each literal **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 4.9 *Let P be a disjunctive revision program. Then R is a P-justified revision of $\emptyset$ if and only if R is an answer set for $dj(P)$.*

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. Disjunctive revision programming is an essential extension of the unitary revision programming. In [MT98] we proved that the existence of P-justified revision problem 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 4.10 *The following problem is Σ_2^P-complete: Given a finite disjunctive revision program and a database I, decide whether 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 Nested expressions in logic programs and revision programs

Lifschitz, Tang and Turner [VLT97] extended the answer set semantics to a class of logic programs with nested expressions permitted in the bodies and heads of rules. In the previous section we extended revision programming to the disjunctive case so that the correspondence with general logic programming be preserved. In this section we will extend the semantics of justified revisions to programs admitting nested occurrences of the connectives. Thus, for instance, expressions of the form $in($**out**$(p)|$**in**$(q))$ will now be allowed as building blocks of program rules.

Our generalization must satisfy criteria similar to those we imposed in Section 4. Specifically, the semantics for programs with nested expressions must coincide with the semantics of justified revisions from Section 4.2 for disjunctive revision programs. Moreover, we want to preserve the shifting property. Finally, we want to make sure that revisions of the empty database coincide with Lifschitz, Tang and Turner semantics for logic programs with nested expressions (via an appropriate translation that does not increase the number of rules). We will now describe a construction that satisfies all these three criteria.

First, we need to define precisely the syntax of nested expressions in revision programming. *Elementary formulas* are revision literals and the symbols $\bot$ ("false") and $\top$ ("true"). Recall that a revision literal is an expression of the form **in**(a) or **out**(a), where a is an atom from U. *Formulas* are built from elementary formulas using the unary connectives in and out and binary connectives "," (conjunction) and "|" (disjunction).

Unary connectives in and out are applied to formulas and should not be confused with **in** and **out** appearing in literals. In particular, the atoms from U are *not* nested expressions.

A *nested revision rule* is an expression of the form $H \leftarrow B$, where H and B are formulas, called the *head* and the *body* of the rule. A *nested revision program* is a set of nested revision rules.

The formulas, rules and programs that do not contain unary operators in and out are called *basic*. We will first define the notion of a model of a basic formula. A set X of literals *satisfies* (is a *model* of) a basic formula F (denoted $X \models F$) if

1. $F = \top$, or
2. F is a literal and $F \in X$, or
3. $F = (A, B)$ for some formulas A and B, $X \models A$ and $X \models B$, or
4. $F = (A|B)$ for some formulas A and B, $X \models A$ or $X \models B$

Let P be a basic revision program. A set of literals $X \subseteq Lit$ is *closed under* P, if for every rule $H \leftarrow B$ in P, $X \models H$ whenever $X \models B$. Next, we define the necessary change entailed by a basic program. Let P be a basic revision program. A set of literals $X \subseteq Lit$ is a *necessary change* of P ($NC(P)$) if it is a minimal (relative to set inclusion) set of revision literals that is closed under P.

Now, we come to the next component of the definition of justified revisions, namely that of the reduct. We will need to reduce both formulas and rules and we will describe reducts with respect to a single database and a pair of databases.

The *reduct* of a formula, rule or program F relative to a set X of atoms (denoted F^X) is defined recursively, as follows (recall that given a set of atoms X, $X^c = \{$**in**$(a) : a \in X\} \cup \{$**out**$(b) : b \notin X\}$):

1. for an elementary F, $F^X = F$.
2. $(F, G)^X = (F^X, G^X)$.
3. $(F|G)^X = (F^X|G^X)$.
4. $(out\ F)^X = \begin{cases} \bot, & \text{if } X^c \models F^X \\ \top, & \text{otherwise} \end{cases}$

5. $(in\ F)^X = \begin{cases} \bot, & \text{if } X^c \not\models F^X \\ \top, & \text{otherwise} \end{cases}$

6. $(F \leftarrow G)^X = F^X \leftarrow G^X$

7. $P^X = \{(F \leftarrow G)^X : F \leftarrow G \in P\}$

Notice that here X is a set of atoms, not of literals.

We are now ready to define the reduct with respect a pair of databases. Let I and R be two databases. Let F be a formula, rule or program. A $reduct$ of F with respect to I, R (denoted $F^{(I,R)}$) is obtained from F^R by replacing each occurrence of literals from $I(I,R)$ by $\top$, and replacing each occurrence of literals from $\{\alpha^D : \alpha \in I(I,R)\}$ by $\bot$.

Definition 5.1 *Let P be a nested revision program. A database R is a P-*justified revision *of a database I if for some necessary change L of $P^{(I,R)}$, L is coherent and $R = I \oplus L$.*

The construction outlined above satisfies the three “benchmarks” that we described above.

Theorem 5.2 *Let P be a disjunctive revision program. Then R is a P-justified revision of I if and only if R is a P-justified revision of I when P is treated as a nested revision program.*

The operator T_W can be extended to rules with nested expressions. Namely, as before, all revision literals **in**(a) and **out**(a), where $a \in W$, that appear in nested expressions H and B forming a revision rule, are replaced by their duals. All other revision literals are left unchanged.

Theorem 5.3 (Shift theorem) *Let I_1 and I_2 be databases (subsets of U), Let P be a nested revision program. Let $W = I_1 \div I_2$. Then, R_1 is P-justified revision of I_1 if and only if $T_W(R_1)$ is $T_W(P)$-justified revision of I_2.*

Given a nested revision program P we define a nested logic program $nlp(P)$ (of the type considered by Lifschitz, Tang and Turner) to be a program obtained from P by replacing each literal $l =$ **in**(a) ($l =$ **out**(a)) by a ($not\ a$), replacing each unary connective out by not, and replacing each unary connective in by $not\ not$.

Theorem 5.4 *Let P be a nested revision program. Then R is a P-justified revision of $\emptyset$ if and only if R is an answer set for $nlp(P)$.*

6 Future work

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, I) must first be compiled into a general logic program (by applying the transformation T_I to P). Then, answer sets to $T_I(P)$ must be computed and “shifted” back by means T_I.

To compute the answer sets of the general logic program $T_I(P)$, one might use any of the existing systems computing stable models of logic programs (for instance s-models [NS96] or DeReS [CMMT95]). 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

[ALP$^+$98] J.J. Alferes, J.A. Leite, L.M. Pereira, H. Przymusinska, and T.C. Przymusinski. Dynamic logic programming. Unpublished manuscript, 1998.

[AP97] J.J. Alferes and L.M. Pereira. Update-programs can update programs. In *Proceedings of the JICSLP-96 Workshop on Non-Monotonic Extensions of Logic Programming: Theory, Applications and Implementations*, pages 110–131. Springer-Verlag, 1997. Lecture Notes in Computer Science 1216.

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

[BM96] N. Bidoit and S. Maabout. Update rule programs related to revision programs. In *Proceedings of the JICSLP-96 Workshop on Non-Monotonic Extensions of Logic Programming: Theory, Applications and Implementations*, 1996.

[CMMT95] P. Cholewiński, W. Marek, A. Mikitiuk, and M. Truszczyński. Experimenting with nonmonotonic reasoning. In *Proceedings of the 12th International Conference on Logic Programming*, pages 267–281. MIT Press, 1995.

[EG95] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: propositional case. *Annals of Mathematics and Artificial Intelligence*, 15:289–324, 1995.

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

[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 revision programs. In *Logics in AI. Proceedings of JELIA '94.* Lecture Notes in Artificial Intelligence. Springer-Verlag, 1994.

[MT95a] 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.

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

[MT95b] N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proceedings of AAAI-95*, pages 1978–1984, 1995.

[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] Teodor C. Przymusinski and Hudson 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 *Proceedings of LPNMR'95*, pages 344–357. Berlin: Springer-Verlag, 1995. Lecture Notes in Computer Science, 928.

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

[VLT97] L. R. Tang V. Lifschitz and H. Turner. Nested expressions in logic programs. unpublished draft, 1997.