ABSTRACT OF DISSERTATION

Inna Valentinovna Pivkina

The Graduate School
University of Kentucky
2001

REVISION PROGRAMMING: A KNOWLEDGE REPRESENTATION FORMALISM

ABSTRACT OF DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
at the University of Kentucky

By

Inna Valentinovna Pivkina

Lexington, Kentucky

Co-Directors:	Dr. Wiktor Marek,	Professor of Computer Science
and	Dr. Mirosław Truszczyński,	Professor of Computer Science

Lexington, Kentucky

2001

ABSTRACT OF DISSERTATION

REVISION PROGRAMMING: A KNOWLEDGE REPRESENTATION FORMALISM

The topic of the dissertation is revision programming. It is a knowledge representation formalism for describing constraints on databases, knowledge bases, and belief sets, and providing a computational mechanism to enforce them. Constraints are represented by sets of revision rules. Revision rules could be quite complex and are usually in a form of conditions (for instance, if these elements are present and those elements are absent, then this element must be absent). In addition to being a logical constraint, a revision rule specify a preferred way to satisfy the constraint. Justified revisions semantics assigns to any database a set (possibly empty) of revisions. Each revision satisfies the constraints, and all deletions and additions of elements in a transition from initial database to the revision are derived from revision rules.

Revision programming and logic programming are closely related. We established an elegant embedding of revision programs into logic programs, which does not increase the size of a program. Initial database is used in transformation of a revision program into the corresponding logic program, but it is not represented in the logic program.

The connection naturally led to extensions of revision programming formalism which correspond to existing extensions of logic programming. More specific, a disjunctive and a nested versions of revision programming were introduced.

Also, we studied annotated revision programs, which allow annotations like confidence factors, multiple experts, etc. Annotations were assumed to be elements of a complete infinitely distributive lattice. We proposed a justified revisions semantics for annotated revision programs which agreed with intuitions.

Next, we introduced definitions of well-founded semantics for revision programming. It assigns to a revision problem a single "intended" model which is computable in polynomial time.

Finally, we extended syntax of revision problems by allowing variables and implemented translators of revision programs into logic programs and a grounder for revision programs. The implementation allows us to compute justified revisions using existing implementations of the stable model semantics for logic programs.

KEYWORDS: Knowledge Representation, Logic Programming, Revision Programming, Constraint Satisfaction, Database Updates.

Inna Valentinovna Pivkina

July 31, 2001

REVISION PROGRAMMING: A KNOWLEDGE REPRESENTATION FORMALISM

By

Inna Valentinovna Pivkina

Dr. Wiktor Marek
Co-Director of Dissertation

Dr. Mirosław Truszczyński
Co-Director of Dissertation

Grzegorz Wasilkowski
Director of Graduate Studies

July 31, 2001

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the dissertation in whole or in part requires also the consent of the Dean of The Graduate School of the University of Kentucky.

DISSERTATION

Inna Valentinovna Pivkina

The Graduate School
University of Kentucky
2001

REVISION PROGRAMMING: A KNOWLEDGE REPRESENTATION FORMALISM

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
at the University of Kentucky

By

Inna Valentinovna Pivkina

Lexington, Kentucky

Co-Directors:	Dr. Wiktor Marek,	Professor of Computer Science
and	Dr. Mirosław Truszczyński,	Professor of Computer Science

Lexington, Kentucky

2001

Table of Contents

List of Tables

List of Figures

List of Files

pivkina.pdf	.pdf	484 kb

Chapter 1

Introduction

Knowledge representation is one of the fundamental topics in Artificial Intelligence. As it is stated in [5], any intelligence should at least involve knowing some things and be able to use them to respond appropriately to a given situation. Therefore, if we want to have an intelligent computer system we must have a way of representing the things it should know (knowledge) in such a way that they can be encoded within the computer system. Examples of knowledge include facts and processes which cause those facts to change. By representation we mean a set of syntactic and semantic conventions that makes it possible to describe things. By *syntax* we mean a set of rules for combining symbols to form valid expressions. By *semantics* we mean the specification of how such expressions are to be interpreted.

The thesis is devoted to one particular knowledge representation formalism, revision programming.

1.1 Introduction to revision programming

Revision programming is a knowledge representation formalism to describe and study the process of database updates and, more generally, principles of change. By database we mean a collection of facts. Revision programming was introduced by Marek and Truszczyński in a series of papers [21], [22], [23]. Databases are represented as collections of data items (*atoms*) from some finite universe, say U. In revision programming updates are specified by means of *revision programs* - sets of *revision rules*, which are expressions of the two types:

$$\mathbf{in}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n) \tag{1.1}$$

or

$$\mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n), \tag{1.2}$$

where a, a_i and b_i are from U. The idea is that $\mathbf{in}(a)$ says the atom a is (or ought to be) in a database, while $\mathbf{out}(a)$ says a is (or ought to be) out.

Revision rules specify logical constraints on databases. For example, the rule (1.1) (called an *in-rule*) 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.

At the same time revision rules express a preferred way to enforce the constraints. For instance, the rule (1.2) (called an *out-rule*) says that in the case when a and all a_i, $1 \leq i \leq m$, are *in* a database, and all b_j, $1 \leq j \leq n$, are *not* in the database, we should remove a from the database rather than remove one of a_i, $1 \leq i \leq m$, or add one of b_j, $1 \leq j \leq n$.

Marek and Truszczyński [23] defined the justified revision semantics for revision programs. Given a revision program P and a database I (*initial* database), 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. Each P-justified revision satisfies all constraints imposed by P ([23]).

Justified revisions can be thought of as generalizations of simple database updates (adding or removing data items to/from a database). Indeed, given a database I, the database $I \cup \{a\}$ is a $\{\mathbf{in}(a) \leftarrow\}$-justified revision of I, while the database $I \setminus \{a\}$ is a $\{\mathbf{out}(a) \leftarrow\}$-justified revision of I. More generally, given a revision program P:

$$\begin{aligned} \mathbf{in}(a_1) &\leftarrow \\ &\ldots \\ \mathbf{in}(a_k) &\leftarrow \\ \mathbf{out}(b_1) &\leftarrow \\ &\ldots \\ \mathbf{out}(b_l) &\leftarrow \quad , \end{aligned}$$

and a database I, if P is not contradictory, that is, $\{a_1, \ldots, a_k\} \cap \{b_1, \ldots, b_l\} = \emptyset$, then the only P-justified revision of I is the database $(I \cup \{a_1, \ldots, a_k\}) \setminus \{b_1, \ldots, b_l\}$. Thus, all simple updates can be represented as justified revisions. At the same time justified revisions can represent more complex updates, like conditional updates or side effects of updates. For instance, if

$$\begin{aligned} P = \{&\mathbf{in}(it\_is\_raining) \leftarrow \quad , \\ &\mathbf{in}(mark\_is\_wet) \leftarrow \mathbf{in}(it\_is\_raining), \mathbf{out}(mark\_has\_umbrella)\}, \end{aligned}$$

then a P-justified revision of a database I will contain data item $mark\_is\_wet$ only if I does not contain data item $mark\_has\_umbrella$.

The intuition behind a P-justified revision of a database is based on the principle of inertia. Inertia expresses "frame axioms" as described in [24]. The principle of inertia says that the status of each element in the database should remain the same unless explicitly changed by the update. Therefore, each insertion or deletion performed in order to convert initial database into revised database must be justified by some rule in the revision program.

The principle of inertia is analogous to the closed-world assumption (CWA) in Artificial Intelligence. Roughly, CWA assumes that every atom that can be false, is false. In Artificial Intelligence, CWA is expressed as a rule schema $\frac{\not\vdash p}{\neg p}$. One can, however, present CWA in a manner very similar to revision programming. Indeed, let I be a database which initially assigns to all atoms logical value "false". We can express provability as rules for changing logical values of atoms. Then, CWA(T), if consistent, describes (a unique) T-justified revision of the initial database.

The revision process specified by revision programming is more complex than the one described above for CWA. It is context-dependent and similar to the one used in stable semantics of logic programming. The concept of justified revision is described in details in Section 2.4.

Revision programming is closely related to logic programming. Logic programming represents knowledge as a set of logical axioms (rules) and then uses an interpreter (a theorem prover) which deduces consequences from it. A logic program rule is of the form

$$L_0 \leftarrow L_1, \ldots, L_n, \tag{1.3}$$

where each L_i is a logical statement. If $L_1, \ldots L_n$ are true then rule 1.3 means that L_0 is also true. Logic programming and the relation of revision programming to it is described in detail in Chapters 2 and 3.

Roots of a recent formalism of dynamic logic programming [1] can be traced back to revision programming. Dynamic logic programming deals with updates of knowledge bases represented by logic programs. It coincides with revision programming when the initial program is just a set of facts ([1]).

1.2 Contribution of the Thesis

One of the key results was the discovery of a complete symmetry between **in**s and **out**s in revision programming, which is expressed in the Shifting Theorem (Theorem 3.2.2). The Shifting Theo-

rem states that changing status of any set of atoms in a revision program (replacing **in**s by **out**s) and changing status of the same atoms in initial and revised databases (adding atoms which were not there and removing atoms which were there) preserves justified revisions. In particular, every revision problem (revision program and initial database) is equivalent to a revision problem with empty database. By equivalent we mean that their justified revisions are in one-to-one correspondence. This allowed us to establish an elegant embedding of revision programs into logic programs that does not increase the size of a program. An initial database is used in the transformation of a revision program into the corresponding logic program, but it is not represented in the logic program.

The embedding naturally led to extensions of revision programming formalism which correspond to existing extensions of logic programming. More specificly, a disjunctive version and a nested version of revision programming were introduced. The Shifting Theorem was also used as a criterion which all versions of revision programming must satisfy.

We also studied annotated revision programs, which allow annotations like confidence factors, multiple experts, etc. Annotations were assumed to be elements of a complete infinitely distributive lattice. We proposed a justified revisions semantics for annotated revision programs which agrees with intuitions.

Next, definitions of well-founded semantics for revision programming were proposed. They assign to a revision problem a single "intended" model which is computable in polynomial time.

Finally, we extended the syntax of revision problems by allowing variables, arithmetic functions and some special constructs, and implemented translators of revision programs into logic programs and a grounder for revision programs. The implementation allows us to compute justified revisions using an existing implementation of the stable model semantics for logic programs, Smodels ([25]).

1.3 Organization

Chapter 2 contains the basic definitions from logic programming and revision programming. In particular, it describes justified revisions semantics for revision programs and an embedding of revision programs into logic programs. Chapter 3 concentrates on embeddings of logic programs

into revision programs. Chapter 4 defines well-founded semantics for revision programs. Chapter 5 discusses extensions of revision programming, in particular, disjunctive and nested revision programs. Chapter 6 presents annotated revision programs. Annotations allow for confidence factors, multiple experts, and so on.

Chapter 2

Preliminaries

This chapter contains basic notions that will be used in the thesis. We start with monotone operators and fixpoints. Then we present Horn programs. Next, we give formal definitions of logic programs and their stable and well-founded semantics. Then we introduce propositional revision programs with justified revision semantics, some of their properties and embedding of logic programs into revision programs. Finally we discuss revision programs with variables and show how their semantics can be obtained from propositional semantics of revision programs.

2.1 Monotone operators and fixpoints.

Consider an arbitrary set U. A set of all subsets of U we denote by $\mathcal{P}(U)$. An *operator* on a set $\mathcal{P}(U)$ is a function from $\mathcal{P}(U)$ to $\mathcal{P}(U)$.

Let T be an operator on $\mathcal{P}(U)$. A set $X \subseteq U$ is a *fixpoint* of T if $T(X) = X$.

A least fixpoint (in $\subseteq$ ordering) of T (if it exists) is denoted $lfp(T)$. A greatest fixpoint of T (if it exists) is denoted $gfp(T)$.

Operator T is *monotone* if for any subsets $X, Y \subseteq U$, $X \subseteq Y$ implies $T(X) \subseteq T(Y)$.

Theorem 2.1 (Knaster-Tarski theorem [30]) *If T is a monotone operator, then T has a least fixpoint and a greatest fixpoint.*

Operator T is *anti-monotone* if for any subsets $X, Y \subseteq U$, $X \subseteq Y$ implies $T(Y) \subseteq T(X)$.

Proposition 2.1 *If T is an anti-monotone operator then*

1. *T^2 is monotone;*
2. *$T(lfp(T^2)) = gfp(T^2)$ and $T(gfp(T^2)) = lfp(T^2)$;*
3. *for any fixpoint X of T, $lfp(T^2) \subseteq X \subseteq gfp(T^2)$.*

If U is finite, then for any anti-monotone operator T, we can compute the least and the greatest fixpoints of operator T^2 by iterating T a finite number of times. Namely, we compute the sets

$T^0(\emptyset) = \emptyset$, $T^n(\emptyset) = T(T^{n-1}(\emptyset))$ for $n = 1, 2, \ldots$, until a value n is found for which $T^n(\emptyset) = T^{n+2}(\emptyset)$. Then, one of the sets $T^n(\emptyset)$, $T^{n+1}(\emptyset)$ is the least fixpoint of T^2, and the other one is the greatest fixpoint of T^2, depending on whether n is even or odd.

2.2 Horn programs.

By an alphabet $\mathcal{A}$ we mean a finite or countably infinite disjoint set of constants, variables, predicate symbols, and function symbols. A *term* over $\mathcal{A}$ is defined recursively as either a variable, a constant or an expression of the form $f(t_1, \ldots, t_n)$, where f is a function symbol of $\mathcal{A}$, and t_i's are terms. An *atom* over $\mathcal{A}$ is an expression of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol of $\mathcal{A}$, and t_i's are terms. A term (resp. atom) is called *ground* if it does not contain variables. The set of ground terms (resp. atoms) of $\mathcal{A}$ is called the *Herbrand universe* (resp. *Herbrand base*) of $\mathcal{A}$.

For the rest of the thesis, except for Section 2.5, we do not consider variables, that is, all atoms are ground atoms. This assumption does not lessen the generality because the semantics of a propositional case can be lifted to a case with variables through grounding. We describe the process in more details in Section 2.5.

Definition 2.1 *A* Horn clause *is an expression of one of the two forms*

$$q \leftarrow p_1, \ldots, p_m \quad (m \geq 0) \tag{2.1}$$

or

$$\leftarrow p_1, \ldots, p_m \quad (m \geq 1) \tag{2.2}$$

where $q, p_1, \ldots, p_m$ *are atoms. Expressions of the first type are called* program clauses *and expressions of the second type are called* goals. *A program clause of the form* $q \leftarrow$ $(m = 0)$ *is called a* fact. $\triangle$

A *Horn theory* is a set of Horn clauses.

A *Horn program* is a set of program clauses.

A set of atoms M is a *model* of (or *satisfies*) a program clause $q \leftarrow p_1, \ldots, p_m$ if $q \in M$ whenever $p_1, \ldots, p_m \in M$. A set of atoms M is a *model* of (or *satisfies*) a goal $\leftarrow p_1, \ldots, p_m$ if $\{p_1, \ldots, p_m\} \not\subseteq M$. A set of atoms M is a *model* of a Horn theory (Horn program) H if M satisfies every Horn clause (program clause) in H.

A Horn theory do not necessarily have a model. However, a Horn program always has a model (for instance, a set of all atoms is a model).

One of the essential properties of Horn programs is that every Horn program has a least model (in $\subseteq$ ordering). It can be computed using the following operator.

Definition 2.2 ([33]) *Given a Horn program P, the* immediate consequence operator T_P *on sets of atoms is defined as follows: $q \in T_P(X)$ if and only if there is a clause $(q \leftarrow p_1, \ldots, p_m) \in P$, such that $p_1, \ldots, p_m \in X$.* $\triangle$

Operator T_P is monotone. By the Knaster-Tarski theorem, it has a least fixpoint. In fact, $lfp(T_P)$ is the least model of P. We denote the least model of a Horn program P by $Least(P)$.

The following lemma shows that a least model of a Horn program does not change if we add to the program rules which are satisfied by the least model.

Lemma 2.1 *Let M be a least model of a Horn program P and a model of a Horn program P'. Then, M is the least model of $P \cup P'$.*

A Horn theory H splits into two disjoint subsets H_g consisting of goals, and H_p consisting of program clauses.

Lemma 2.2 *Let H be a Horn theory, $H = H_g \cup H_p$ its decomposition into goal part and program part. Then,*

1. *M is a model of H if and only if M is a model of H_p and satisfies all goals from H_g;*
2. *H has a model if and only if $Least(H_p)$ satisfies goals from H_g.*

2.3 Logic programs.

Let U be a finite or countably infinite *universe*. Elements of U are called *atoms*.

Definition 2.3 *A* logic program clause *is an expression of the form*

$$p \leftarrow q_1, \ldots, q_m, not\ s_1, \ldots, not\ s_n, \tag{2.3}$$

where $p,\ q_1, \ldots,\ q_m,\ s_1, \ldots,\ s_n$ are atoms. $\triangle$

For a set of atoms $A \subseteq U$, let us denote the set $\{not\ a : a \in A\}$ by $not(A)$. Then, any logic program clause can be represented as $Head \leftarrow Pos \cup not(Neg)$, for some finite sets of atoms Pos and Neg, and an atom $Head$.

Definition 2.4 *A* logic program *is a set of logic program clauses.* $\triangle$

A set $M \subseteq U$ is a *model* of (or *satisfies*) a logic program clause of the form (2.3) if $\{q_1, \ldots, q_m\} \subseteq M$ and $\{s_1, \ldots, s_n\} \cap M = \emptyset$ imply that $p \in M$. A set M is a *model* of (or *satisfies*) a logic program P if M is a model of every clause $r \in P$.

2.3.1 Stable model semantics.

Stable model semantics was first introduced in [10].

Definition 2.5 ([10]) *Let P be a logic program. The reduct of P relative to M, P^M, is obtained from P by*

1. *removing all clauses which contain "$not\ q$" such that q is true in M,*
2. *deleting all negative premises "$not\ q$" (for all $q \in U$) from the remaining clauses.* $\triangle$

P^M is a Horn program. It has a unique least model $Least(P^M)$.

Definition 2.6 ([10]) *M is a* stable model *of P if $M = Least(P^M)$.* $\triangle$

Theorem 2.2 ([10]) *If M is a stable model of P, then M is a model of P.*

Theorem 2.3 ([20]) *The problem of existence of a stable model for a finitie logic program is NP-complete.*

Stable model semantics has some important drawbacks. It is not *universal*, which means that that not every logic program has a stable model. For example, the logic program $P = \{a \leftarrow not\ a\}$ has no stable models. Also, even for finite propositional logic programs the stable semantics can not be effectively computed ([12], [20]).

2.3.2 Well-founded semantics.

The stable model semantics does not fit into a standard paradigm of logic programming languages. While standard approaches assign to a logic program a single "intended" model, stable model semantics assigns to a program a family (possibly empty) of "intended" models.

Proposals to address the issue for stable model semantics were of two types. Proposals of the first type attempt to salvage the notion of a single intended model at a cost of narrowing down the class of programs or weakening the semantics. Apt, Blair and Walker [4] introduced the notion of *stratification*, a syntactic restriction on logic programs with negation. They assigned to each stratified program a single intended model, a perfect model.

Proposal of the second type for stable model semantics was made by van Gelder, Ross and Schlipf ([34], [35]). They assigned to an arbitrary logic program a single intended 3-valued model, a *well-founded model*. In this model, any logic program partitions the set of atoms into three groups: the well-founded atoms, the unfounded atoms, and the rest.

Definition 2.7 *For any program P, the operator γ_P is defined by the equation*

$$\gamma_P(X) = Least(P^X). \qquad \triangle$$

The operator γ_P is anti-monotone. By Proposition 2.1, the operator γ_P^2 is monotone, and has a least fixpoint and a greatest fixpoint.

The well-founded semantics of a logic program is defined as follows.

Definition 2.8 *The atoms that belong to the least fixpoint of γ_P^2 are* well-founded *relative to P. The atoms that do not belong to the greatest fixpoint of γ_P^2 are* unfounded *relative to P. The remaining atoms are called* unknown. $\triangle$

Proposition 2.2 ([14]) *Any stable model for a logic program P*

- *includes all atoms that are well-founded relative to P, and*
- *includes no atoms unfounded relative to P.*

Theorem 2.4 ([34]) *If every atom is either well-founded or unfounded relative to a logic program P then the set of well-founded atoms is the only stable model for P.*

An important feature of the well-founded semantics is that it is computable in polynomial time, in some cases even in linear time ([6]).

2.4 Revision programs and their justified revisions semantics.

In this section we present results from [23, 22] which are used in this thesis. In Section 2.4.1 we present formal definitions of revision programs with justified revisions semantics. In Section 2.4.2 we list basic properties of revision programs. Finally, in Section 2.4.3, we present a natural embedding of logic programs into revision programs.

2.4.1 Definition

Elements of some finite universe U are called *atoms*. Subsets of U are called *databases*. Expressions of the form **in**(a) or **out**(a), where a is an atom, are called *revision literals*. Revision literals will be denoted by greek letters α, etc. For a revision literal **in**(a), its *dual* is the revision literal **out**(a). Similarly, the *dual* of **out**(a) is **in**(a). The dual of a revision literal α is denoted by α^D.

For any set of atoms $B \subseteq U$, we define

$$B^c = \{\textbf{in}(a) : a \in B\} \cup \{\textbf{out}(a) : a \notin B\}.$$

We can think of B^c as a complete representation of B since for every atom $a \in U$, B^c shows whether a is in B or not.

For any set of literals L, we define

$$L^+ = \{a \in U : \textbf{in}(a) \in L\}, \quad L^- = \{a \in U : \textbf{out}(a) \in L\}.$$

Definition 2.9 *A set of revision literals L is* coherent *if it does not contain a pair of dual literals, that is, $L^+ \cap L^- = \emptyset$.* $\triangle$

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

$$I \oplus L = (I \setminus L^-) \cup L^+.$$

Notice that if L is coherent, then $(I \setminus L^-) \cup L^+ = (I \cup L^+) \setminus L^-$.

Definition 2.10 *A* revision rule *is an expression of one of the following two types:*

$$\textbf{in}(a) \leftarrow \textbf{in}(a_1), \ldots, \textbf{in}(a_m), \textbf{out}(b_1), \ldots, \textbf{out}(b_n) \tag{2.4}$$

or

$$\textbf{out}(a) \leftarrow \textbf{in}(a_1), \ldots, \textbf{in}(a_m), \textbf{out}(b_1), \ldots, \textbf{out}(b_n), \tag{2.5}$$

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 (2.4) 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 (imperative) 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 (2.4), the item a must be added to the database (removed from it, in the case of the constraint (2.5)), rather than some item a_i removed or some item b_j added.

If a revision rule r is of the type (2.4), then the *head* of r, denoted $head(r)$, is the revision literal $\textbf{in}(a)$. If r is of the type (2.5), then the *head* of r, denoted $head(r)$, is the revision literal $out(a)$. If r is a revision rule of the type (2.4) or (2.5), then *body* of r is the set $body(r) = \{\textbf{in}(a_1), \ldots, \textbf{in}(a_m), \textbf{out}(b_1), \ldots, \textbf{out}(b_n)\}$.

Definition 2.11 *A* revision program *is a collection of revision rules.* △

Definition 2.12 *A set of atoms $B \subseteq U$ is a* model *of (or* satisfies*) a revision literal* $\textbf{in}(a)$ *(resp.,* $\textbf{out}(a)$*), if $a \in B$ (resp., $a \notin B$). A set of atoms B is a* model *of (or* satisfies*) a revision rule r if either B is not a model of at least one revision literal from the body of r, or B is a model of $head(r)$. A set of atoms B is a* model *of (or* satisfies*) a revision program P if B is a model of every rule in P.* △

Let P be a revision program. The *size* of P (denoted $size(P)$) is the number of occurrences of atoms in P. A set of all revision literals appearing in P is denoted $var(P)$. A set of the heads of all rules in P is denoted by $head(P)$.

Definition 2.13 *Let P be a revision program. The* necessary change *of P, $NC(P)$, is the least model of P, when treated as a Horn program built of independent propositional atoms of the form* **in**(a) *and* **out**(b). $\triangle$

Example 2.1 *Let a revision program P be $\{in(a) \leftarrow,\ \ out(b) \leftarrow in(a),\ \ out(c) \leftarrow out(a)\}$. Then, $NC(P) = \{in(a), out(b)\}$.*

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

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

Definition 2.14 *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 revision literals in $I(I, R)$.* $\triangle$

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.

Definition 2.15 *The database R is a P-*justified revision *of I if the necessary change of $P_{I,R}$ is coherent and if $R = I \oplus NC(P_{I,R})$.* $\triangle$

The following example illustrates the notion of justified revision.

Example 2.2 Assume that we need to form a committee. There are four people which can be on the committee: Ann, Bob, Chris 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, Chris is an expert from another country and does not speak English well enough yet. So, if Chris is on the committee, David should be on the committee too, because David can be an interpreter. If David is not on the committee, Chris 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 Chris 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 $I = \{Ann, Chris\}$ with respect to revision program P:

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

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

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

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

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

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

2.4.2 Basic properties

There is an alternative definition of P-justified revisions also presented in [23]. It is based on a different notion of a reduct – a counterpart of the Gelfond-Lifschitz reduct in logic programming (Definition 2.5).

Definition 2.16 ([23]) *Let P be a revision program, and let I and R be two databases. The* GL-reduct *of P with respect to (I, R) (denoted $P_R|I$) is obtained from P by*

1. *eliminating from P every rule whose body is not satisfied by R (the resulting program is denoted by P_R),*
2. *eliminating each literal that is satisfied by I from the body of each rule in P_R.* △

Each of the reducts, $P_{I,R}$ and $P_R|I$, can be used to define the notion of P-justified revision, as the following theorem shows.

Theorem 2.5 ([23]) *Let P be a revision program and let I and R be two databases. The following two conditions are equivalent:*

- *$NC(P_{I,R})$ is coherent and $R = I \oplus NC(P_{I,R})$,*
- *$NC(P_R|I)$ is coherent and $R = I \oplus NC(P_R|I)$.*

For justified revisions, the necessary changes of both reducts are the same:

Theorem 2.6 ([23]) *Let P be a revision program and R be a P-justified revision of I. Then, $NC(P_{I,R}) = NC(P_R|I) = head(P_R)$.*

In the thesis we will frequently use the following characterizations also given in [23].

Theorem 2.7 ([23]) *The following conditions are equivalent:*
1. A database R is a P-justified revision of a database I,
2. $NC(P \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}) = R^c$;
3. $NC(P_{I,R}) \cup I(I, R) = R^c$.

One of the fundamental properties of P-justified revisions is that they indeed are models of P.

Theorem 2.8 ([23]) *Let P be a revision program and let I be a database. If a database R is a P-justified revision of I, then R is a model of P.*

The notion of a justified revision also satisfies a minimality principle. Namely, justified revisions of a database differ from the database by as little as possible. To describe how much databases differ from each other we use the notion of a symmetric difference. Given two databases I and J, the *symmetric difference* of I and J is defined as $I \div J = (I \setminus J) \cup (J \setminus I)$.

Theorem 2.9 ([23]) *Let P be a revision program and let I be a database. If R is a P-justified revision of I, then $R \div I$ is minimal in the family $\{B \div I : B$ is a model of $P\}$.*

The following theorem demonstrates that "additional evidence does not destroy justified revisions". If we add to a revision program P some rules that are already satisfied by a P-justified revision, then there is no need to change the revision.

Theorem 2.10 ([23]) *Let R be a P-justified revision of I. Assume that P' is a revision programs such that R is a model of P'. Then, R is a $(P \cup P')$-justified revision of I.*

The next theorem shows that if the current database satisfies the revision program, then no change is justified.

Theorem 2.11 ([23]) *If a database I satisfies a revision program P then I is a unique P-justified revision of I.*

Given a revision program P, the *dual* of P (P^D in symbols) is the revision program obtained from P by simultaneously replacing all occurrences of all revision literals by their duals. The duality theorem states that revision programs P and P^D are, in a sense, equivalent.

Theorem 2.12 (Duality Theorem [23]) *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 $U \setminus R$ is a P^D-justified revision of $U \setminus I$.*

The problem of existence of justified revisions is NP-complete.

Theorem 2.13 ([23]) *The following problem is NP-complete: Given a finite revision program P and a finite database I, decide whether I has a P-justified revision.*

2.4.3 Embedding of logic programs into revision programs.

The following interpretation of logic programs as revision programs was proposed in [23].

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

Under this interpretation, models and stable models of logic programs can be represented as models and justified revisions of revision programs.

Theorem 2.14 ([23]) *Let P be a logic program.*

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

Therefore, logic programs can be viewed as special revision programs. Revision programs, in turn, can be represented by means of logic programs. We discuss this in detail in Chapter 3.

2.5 Revision programs with variables

In order to use revision programs to solve more than one instance of a problem and use it as a programming language, we should be able to use variables.

Although most of the thesis deals with propositional case of revision programming, we can extend the syntax of revision programming by allowing variables and predicates as in the Datalog syntax ([32]). Semantics for revision programming with variables can be obtained by lifting propositional semantics as follows.

We use the same notions of alphabet, term, atom, Herbrand universe, and Herbrand base as defined in Section 2.2. We use the same definitions of revision literals, revision rules, and revision programs as in Section 2.4.1, only now atoms may contain variables.

An *instance* of an atom is constructed by replacing variables in the atom by ground terms in a systematic way (same variables are replaced by the same terms). Similarly, an *instance* of a revision literal (a revision rule) is constructed by replacing variables in the revision literal (the revision rule) by ground terms in a systematic way. The *Herbrand instantiation* of a revision program P (denoted $HI(P)$) is the set of all ground instances of the revision rules of the revision program that may be constructed using terms in the Herbrand universe.

We can also use variables in initial databases. We define a database to be a set of atoms which may contain variables. Then, the *Herbrand instantiation* of a database I (denoted $HI(I)$) is the set of all ground instances of atoms of the initial database.

Now we can define a notion of a justified revision as follows.

Definition 2.17 *Let P be a revision program and I be an initial database (P and I may contain variables). A database R is a P-*justified revision *of I if R is a $HI(P)$-justified revision of $HI(I)$.* $\triangle$

Example 2.3 *Let an alphabet $\mathcal{A}$ consist of constants $\{1, 2\}$, variable $\{X\}$, and unary predicate symbols $\{red, blue\}$. Let a revision program P be*

$$\begin{aligned} \mathbf{out}(blue(X)) &\leftarrow \mathbf{in}(red(X)) \\ \mathbf{out}(red(X)) &\leftarrow \mathbf{in}(blue(X)). \end{aligned}$$

Let $I = \{red(X), blue(X)\}$.

Then, the Herbrand universe is $\{1, 2\}$, the Herbrand base is $\{red(1), red(2), blue(1), blue(2)\}$. The Herbrand instantiation of P is

$$\begin{aligned} \mathbf{out}(blue(1)) &\leftarrow \mathbf{in}(red(1)) \\ \mathbf{out}(blue(2)) &\leftarrow \mathbf{in}(red(2)) \\ \mathbf{out}(red(1)) &\leftarrow \mathbf{in}(blue(1)) \\ \mathbf{out}(red(2)) &\leftarrow \mathbf{in}(blue(2)). \end{aligned}$$

The Herbrand instantiation of I is $\{red(1), red(2), blue(1), blue(2)\}$.

Therefore, P-justified revisions of I are $R_1 = \{red(1), blue(2)\}$, and $R_2 = \{blue(1), red(2)\}$.

Chapter 3

Translation of Revision Programming into Logic Programming

Revision programming and logic programming are closely related. Section 2.4.3 described an embedding of logic programs into revision programs. This chapter is devoted to embeddings of revision programs into logic programs. Section 3.1 contains the Przymusinski–Turner encoding of revision programs as logic programs ([26]). Their encoding includes an explicit representation of an initial database as part of the logic program. Consequently, the size of the encoding is always greater than the size of the original revision program. A more elegant encoding, which does not increase the size of a program, was found and published in [18]. This encoding is presented in Section 3.2.

3.1 Explicit representation of initial database

The details of the Przymusinski–Turner encoding of revision programs as logic programs are given in Section 3.1.1. In this encoding Przymusinski and Turner represent an initial database as a set of facts. Their result can be generalized to allow not only facts but any logic program rules in the description of an initial database. The generalization is presented in Section 3.1.2.

3.1.1 Przymusinski–Turner translation

In order to translate a revision program and an initial database into a logic program Przymusinski and Turner introduce additional atoms $\mathbf{in}_I(q)$, $\mathbf{out}_I(q)$ to represent the initial database. The intuitive meaning of atoms $\mathbf{in}_I(q)$ and $\mathbf{out}_I(q)$ is 'q is in the initial database' and 'q is not in the initial database'. They consider a propositional language $\mathcal{K}$ with the set of propositional letters consisting of $\{\mathbf{in}(q) : q \in U\} \cup \{\mathbf{out}(q) : q \in U\} \cup \{\mathbf{in}_I(q) : q \in U\} \cup \{\mathbf{out}_I(q) : q \in U\}$.

The Przymusinski–Turner translation of a revision program RP and an initial database B_I into a logic program is as follows.

Definition 3.1 ([26]) The translation of the revision program RP and the initial database B_I into a logic program *is defined as the logic program* $\mathcal{P}(RP, B_I) = P_I \cup P_N \cup RP$ *over* $\mathcal{K}$ *consisting of the following three subprograms:*

Initial Knowledge Rules P_I*: All atoms* $q \in B_I$ *are initially in and all atoms* $s \notin B_I$ *are initially out:*

$$P_I = \{\mathbf{in}_I(q) \leftarrow : \ q \in B_I\} \cup \{\mathbf{out}_I(s) \leftarrow : \ s \notin B_I\};$$

Inertia Rules P_N*: If* q *was initially in (respectively, out) then after revision it remains in (respectively, out) unless it was forced out (respectively, in):*

$$P_N = \{\mathbf{in}(q) \leftarrow \mathbf{in}_I(q), not\ \mathbf{out}(q) \ ; \ \mathbf{out}(q) \leftarrow \mathbf{out}_I(q), not\ \mathbf{in}(q) \ : \ q \in U\};$$

Revision Rules RP*: All the revision rules that belong to the original revision program* RP. $\triangle$

Definition 3.2 ([26]) *A stable model* M *of* $\mathcal{P}(RP, B_I)$ *is called* coherent *if it does not contain both* $\mathbf{in}(q)$ *and* $\mathbf{out}(q)$*, for any* $q \in U$. $\triangle$

Notice that Definition 3.2 is the same as Definition 2.9 except that it refers to different objects. Definition 2.9 is for sets of revision literals (expressions of the form $\mathbf{in}(q)$, $\mathbf{out}(q)$), whereas Definition 3.2 deals with sets of expressions of the form $\mathbf{in}(q)$, $\mathbf{out}(q)$, $\mathbf{in}_I(q)$, and $\mathbf{out}_I(q)$.

The translation $\mathcal{P}(RP, B_I)$ contains the original revision program RP, a complete explicit representation of the initial database as a set of facts, and inertia rules that specify that atoms do not change their status unless they are forced to.

The translation defines an embedding of revision programming into logic programming with stable model semantics.

Theorem 3.1 (Przymusinski and Turner [26]) *Let* RP *be a revision program and* B_I *be an initial database. There is a one-to-one correspondence between* RP*-justified revisions of* B_I *and coherent stable models of its translation* $\mathcal{P}(RP, B_I)$ *into a logic program.*

More precisely, to every RP*-justified revision* B_R *of* B_I *there corresponds a unique coherent stable model* M *of* $\mathcal{P}(RP, B_I)$ *such that:*

$$B_R = \{q : \mathbf{in}(q) \in M\}, \qquad U \setminus B_R = \{q : \mathbf{out}(q) \in M\},$$

and, conversely, for each coherent stable model M *of* $\mathcal{P}(RP, B_I)$ *the set* $B_R = \{q : \mathbf{in}(q) \in M\}$ *is an* RP*-justified revision of* B_I.

The following example illustrates the embedding.

Example 3.1 Let us apply the Przymusinski–Turner translation to the revision program P and the initial database I from Example 2.2. The logic program $\mathcal{P}(P, I)$ is

$$
\begin{aligned}
\mathbf{in}_I(Ann) &\leftarrow \\
\mathbf{out}_I(Bob) &\leftarrow \\
\mathbf{in}_I(Chris) &\leftarrow \\
\mathbf{out}_I(David) &\leftarrow \\
\mathbf{in}(Ann) &\leftarrow \mathbf{in}_I(Ann), not\ \mathbf{out}(Ann) \\
\mathbf{out}(Ann) &\leftarrow \mathbf{out}_I(Ann), not\ \mathbf{in}(Ann) \\
\mathbf{in}(Bob) &\leftarrow \mathbf{in}_I(Bob), not\ \mathbf{out}(Bob) \\
\mathbf{out}(Bob) &\leftarrow \mathbf{out}_I(Bob), not\ \mathbf{in}(Bob) \\
\mathbf{in}(Chris) &\leftarrow \mathbf{in}_I(Chris), not\ \mathbf{out}(Chris) \\
\mathbf{out}(Chris) &\leftarrow \mathbf{out}_I(Chris), not\ \mathbf{in}(Chris) \\
\mathbf{in}(David) &\leftarrow \mathbf{in}_I(David), not\ \mathbf{out}(David) \\
\mathbf{out}(David) &\leftarrow \mathbf{out}_I(David), not\ \mathbf{in}(David) \\
\mathbf{in}(Bob) &\leftarrow \mathbf{out}(Ann) \\
\mathbf{in}(Ann) &\leftarrow \mathbf{out}(Bob) \\
\mathbf{in}(David) &\leftarrow \mathbf{in}(Chris) \\
\mathbf{out}(Chris) &\leftarrow \mathbf{out}(David) \\
\mathbf{out}(Ann) &\leftarrow \mathbf{in}(David) \\
\mathbf{out}(David) &\leftarrow \mathbf{in}(Bob)
\end{aligned}
$$

It is easy to see that $\{\mathbf{in}_I(Ann), \mathbf{out}_I(Bob), \mathbf{in}_I(Chris), \mathbf{out}_I(David), \mathbf{in}(Ann), \mathbf{out}(Bob), \mathbf{out}(Chris)$ $\mathbf{out}(David)\}$ is the only coherent stable model of $\mathcal{P}(P, I)$. It corresponds to $R = \{Ann\}$, which is the only P-justified revision of I as we saw in Example 2.2.

Let us summarize some features of the Przymusinski–Turner translation in the following remark.

Remark 3.1 *Let RP be a revision program. Let B_I be an initial database. Logic program $\mathcal{P}(RP, B_I)$*

1. *has additional atoms to represent B_I (the language $\mathcal{K}$ has $4 * |U|$ propositional letters);*
2. *contains an explicit representation of the initial database and inertia rules, and thus, has $3 * |U|$ more rules than RP, and $size(\mathcal{P}(RP, B_I)) = size(RP) + 7 * |U|$;*
3. *its stable models contain a complete representation of B_I, that is, contain $|U|$ additional elements that are irrelevant to justified revisions.*

These properties of the Przymusinski–Turner translation may be considered to be drawbacks of the embedding, because the problem of finding justified revisions is reduced to a problem of finding stable models of a bigger logic program over a larger language and removing irrelevant elements from the result.

3.1.2 Generalization of Przymusinski and Turner's result

The Przymusinski–Turner embedding represents an initial database as a collection of facts stating for each atom from the universe whether the atom is *in* or *out* of the initial database. Their result can be generalized. Namely, arbitrary logic programs, not only sets of facts, can be used to represent initial databases. The initial database needs to be a stable model of the logic program that is used to represent it. The generalization deals with a description of an initial database. It does not handle program updates in the sense of [2]. Let us now formally present the generalization.

Let RP be a revision program. Let LP be any logic program over $\{\mathbf{in}_I(A) : A \in U\}$.

Given RP and LP, let us define the logic program $\mathcal{P}$ over $\mathcal{K}$ to be the union of *initial program* P_I, *inertia rules* P_N, and revision program RP, where

$$P_I = LP \cup \{\mathbf{out}_I(A) \leftarrow not\ \mathbf{in}_I(A) : A \in U\},\ \text{and}$$

$$P_N = \{\mathbf{in}(A) \leftarrow \mathbf{in}_I(A), not\ \mathbf{out}(A)\ \ ;\ \ \mathbf{out}(A) \leftarrow \mathbf{out}_I(A), not\ \mathbf{in}(A)\ :\ A \in U\}.$$

That is, $\mathcal{P} = P_I \cup P_N \cup RP$. We will say that a stable model of $\mathcal{P}$ is *coherent* if it is coherent in the sense of Definition 3.2 and it does not contain atoms $\mathbf{in}_I(A)$ and $\mathbf{out}_I(A)$ for any $A \in U$.

The following two theorems show that there is a one-to-one correspondence between coherent stable models of $\mathcal{P}$ and RP-justified revisions of stable models of LP.

Theorem 3.2 *Let M be a coherent stable model of $\mathcal{P}$. Then there exists a stable model M_0 of LP, such that $B_R = \{q : \mathbf{in}(q) \in M\}$ is an RP-justified revision of $B_I = \{q : \mathbf{in}_I(q) \in M_0\}$.*

Proof.

By the definition of a stable model, M is the least model of $Q = \mathcal{P}^M$. Let $B_I = \{q : \mathbf{in}_I(q) \in M\}$. Let $B'_I = \{q : \mathbf{out}_I(q) \in M\}$. Observe that $B'_I = U \setminus B_I$. Indeed, since M is coherent, B_I and B'_I are disjoint. Assume that there is an atom $q \in U$ such that $q \notin B_I$. Then, the clause $\mathbf{out}_I(q) \leftarrow$ belongs to $(P_I)^M$. Since M is the least model of

$$Q = \mathcal{P}^M = (P_I)^M \cup (P_N \cup RP)^M,$$

the atom $\mathbf{out}_I(q)$ must belong to M. That is, $q \in B'_I$. Therefore, $B'_I = U \setminus B_I$.

Let $M_0 = M \cap \{\mathbf{in}_I(q) : q \in U\}$. We will show that M_0 is a stable model of LP. Indeed, LP contains only atoms of the form $\mathbf{in}_I(q)$, $q \in U$. Hence,

$$LP^{M_0} = LP^M.$$

Since $LP \subseteq \mathcal{P}$, we have $LP^{M_0} \subseteq Q$. Thus, M is the least model of

$$Q = LP^{M_0} \cup (Q \setminus LP^{M_0}).$$

Let $U_I = \{\mathbf{in}_I(q) : q \in U\}$. All atoms occurring in LP belong to U_I. None of the atoms from U_I appears in the heads of the clauses from $(Q \setminus LP^{M_0})$. Hence, $M_0 = M \cap U_I$ is the least model of LP^{M_0}. Therefore, M_0 is a stable model of LP.

The rest of the proof repeats the proof of Przymusinski and Turner result ([26]). Let us define $B'_R = \{q : \mathbf{out}(q) \in M\}$. The reduct $(P_N)^M$ consists of clauses:

$$\mathbf{in}(q) \leftarrow \mathbf{in}_I(q), \text{ for all } q \notin B'_R$$

$$\mathbf{out}(q) \leftarrow \mathbf{out}_I(q), \text{ for all } q \notin B_R.$$

It can be shown that $B'_R = U \setminus B_R$.

M is also the least model of a modified program obtained by removing some premises which are true in M. Therefore, we can further reduce the reduct $(P_N)^M$ of the set of inertia rules to the set $\widehat{(P_N)^M}$ of all clauses of the form:

$$\mathbf{in}(q) \leftarrow , \text{ for all } q \in B_R \cap B_I, \text{ and}$$

$$\mathbf{out}(q) \leftarrow , \text{ for all } q \in B'_R \cap B'_I.$$

We now show that B_R is an RP-justified revision of B_I. The rules from the program RP are positive. Hence, M is the least model of

$$(P_I)^M \cup \widehat{(P_N)^M} \cup RP.$$

In order to compute the reduct RP_{B_I,B_R} of RP we have to remove from the body of each revision rule all atoms $\mathbf{in}(q)$ such that $q \in B_I \cap B_R$, and all atoms $\mathbf{out}(q)$ such that $q \in B'_R \cap B'_I$. These are precisely the atoms that must be true in M due to the rules $\widehat{(P_N)^M}$. Consequently, M remains the least model of the reduced program

$$(P_I)^M \cup \widehat{(P_N)^M} \cup RP_{B_I,B_R}.$$

The necessary change $NC(RP_{B_I,B_R})$ is the least model of RP_{B_I,B_R}. Let $I = \{q : \mathbf{in}(q) \in NC(RP_{B_I,B_R})\}$ and $O = \{q : \mathbf{out}(q) \in NC(RP_{B_I,B_R})\}$. The program

$$(P_I)^M \cup \widehat{(P_N)^M} \cup RP_{B_I,B_R}$$

consists of three independent parts, where the reduct RP_{B_I,B_R} no longer contains any premises from the other two parts. Consequently, the set of atoms in M consists of:

- $\{\mathbf{in}_I(q) : q \in B_I\} \cup \{\mathbf{out}_I(q) : q \in B'_I\}$,
- $\{\mathbf{in}(q) : q \in B_R \cap B_I\} \cup \{\mathbf{out}(q) : q \in B'_R \cap B'_I\}$,
- $\{\mathbf{in}(q) : q \in I\} \cup \{\mathbf{out}(q) : q \in O\}$.

We conclude that:

$$B_R = \{q : \mathbf{in}(q) \in M\} = (B_R \cap B_I) \cup I,$$

$$B'_R = U \setminus B_R = \{q : \mathbf{out}(q) \in M\} = (B'_R \cap B'_I) \cup O.$$

The necessary change $NC(RP_{B_I,B_R})$ is coherent. Also, $B_R = (B_I \cup I) \setminus O$. Hence, B_R is an RP-justified revision of B_I. □

Theorem 3.3 *Let M_0 be a stable model of LP, $B_I = \{q : \mathbf{in}_I(q) \in M_0\}$. Let B_R be an RP-justified revision of B_I. Then there exists a coherent stable model M of $\mathcal{P}$, such that $B_R = \{q : \mathbf{in}(q) \in M\}$.*

Proof.

Let M_0 be a stable model of LP. Then, $M_I = M_0 \cup \{\mathbf{out}_I(q) : \mathbf{in}_I(q) \notin M_0\}$ is a coherent stable model of P_I. Indeed, since LP does not contain atoms of the form $\mathbf{out}_I(q)$,

$$LP^{M_I} = P^{M_0}.$$

Also, we have that

$$\{\mathbf{out}_I(q) \leftarrow not\ \mathbf{in}_I(q) : q \in U\}^{M_I} \ = \ \{\mathbf{out}_I(q) \leftarrow : \mathbf{in}_I(q) \notin M_0\}.$$

Therefore, M_I is the least model of $(P_I)^{M_I}$. Hence, M_I is a stable model of P_I. Notice that M_I is coherent and for any $q \in U$ either $\mathbf{in}_I(q) \in M_I$ or $\mathbf{out}_I(q) \in M_I$.

Let B_R be an RP-justified revision of B_I. Theorem 2.7 implies

$$NC(RP \cup \{\alpha \leftarrow : \alpha \in I(B_I, B_R)\}) \ = \ \{\mathbf{in}(q) : q \in B_R\} \cup \{\mathbf{out}(q) : q \notin B_R\}.$$

Let $M = M_I \cup \{\mathbf{in}(q) : q \in B_R\} \cup \{\mathbf{out}(q) : q \notin B_R\}$. We need to show that M is a stable model of $\mathcal{P}$. The reduct $\mathcal{P}^M$ is equal to

$$(P_I)^{M_I} \cup (P_N)^M \cup RP.$$

The least model of $\mathcal{P}^M$ must contain M_I - the least model of $(P_I)^{M_I}$. Therefore, the least model of $\mathcal{P}^M$ is also the least model of the program

$$(P_I)^{M_I} \cup Q \cup RP,$$

where $Q = \{\mathbf{in}(q) \leftarrow : q \in B_R \cap B_I\} \cup \{\mathbf{out}(q) \leftarrow : q \notin B_R \cup B_I\}$ is obtained from $(P_N)^M$ by removing some premises which are true in the least model. Notice, that $Q = \{\alpha \leftarrow : \alpha \in I(B_I, B_R)\})$. Next, observe that the programs $(P_I)^{M_I}$ and $Q \cup RP$ contain no common atoms. Therefore, the least model of

$$(P_I)^{M_I} \cup Q \cup RP$$

is the union of the least model of $(P_I)^{M_I}$ (which is M_I) and the least model of $Q \cup RP$ (which is $\{\mathbf{in}(q) : q \in B_R\} \cup \{\mathbf{out}(q) : q \notin B_R\}$). In other words, the least model of $\mathcal{P}^M$ is M. Hence, M is a stable model of $\mathcal{P}$. □

Remark 3.2 *Theorem 3.2 and Theorem 3.3 generalize Przymusinski and Turner's result. Indeed, let RP be a revision program, and let B_I be an initial database. Let LP be the logic program over $\{\mathbf{in}_I(q) : q \in U\}$ such that $LP = \{\mathbf{in}_I(q) \leftarrow : q \in B_I\}$. The only stable model of P is the set $\{\mathbf{in}_I(q) : q \in B_I\}$. Thus, applying Theorem 3.2 and Theorem 3.3 to LP and RP we get the statement of Theorem 3.1.*

3.1.3 Computing justified revisions

One of the methods to compute justified revisions is to reduce them to logic programs and use techniques for computing stable models. In particular, we can use the Przymusinski-Turner translation to embed revision programs into logic programs.

We use the following slight modification of the translation which allows us to reduce the number of rules in the obtained logic program.

Definition 3.3 (modification of Definition 3.1) The translation of a revision program P and an initial database I into a logic program *is defined as the logic program $\mathcal{P}'(P, I) = P'_N \cup RP$ consisting of the following two subprograms:*

Inertia Rules *P'_N: If q was initially in (respectively, out) then after revision it remains in (respectively, out) unless it was forced out (respectively, in):*

$$P'_N = \{\mathbf{in}(q) \leftarrow not\ \mathbf{out}(q)\ :\ q \in I\} \cup \{\mathbf{out}(q) \leftarrow not\ \mathbf{in}(q)\ :\ q \notin I\};$$

Revision Rules *P: All the revision rules that belong to the original revision program P.* $\triangle$

The modification only removes the unnecessary representation of the initial database in the translation.

Lemma 3.1 *Let P be a revision program, I be a database. Let $\mathcal{P}(P, I)$ be the translation of P and I into a logic program according to Definition 3.1. Let $\mathcal{P}'(P, I)$ be the translation of P and I into a logic program according to Definition 3.3. Then, M is a stable model of $\mathcal{P}'(P, I)$ if and only if $M \cup \{\mathbf{in}_I(q) : q \in I\} \cup \{\mathbf{out}_I(q) : q \notin I\}$ is a stable model of $\mathcal{P}(P, I)$.*

This translation allows us to compute justified revisions according to the following scheme.

1. Given a revision program P and initial database I, compute the logic program $\mathcal{P}'(P, I)$ by adding to P inertia rules P'_N.

2. Compute stable models of $\mathcal{P}'(P, I)$.

3. P-justified revisions of I are obtained from coherent stable models of $\mathcal{P}'(P, I)$ via the following formula: $R = \{a : \mathbf{in}(a) \in M\}$, where M is a coherent stable model of $\mathcal{P}'(P, I)$, and R is a P-justified revision of I.

The following example illustrates the above scheme.

Example 3.2 *Let us apply the translation from Definition 3.3 to the revision program P and the initial database I from Example 2.2. The logic program $\mathcal{P}'(P, I)$ is*

$$
\begin{aligned}
\mathbf{in}(Ann) &\leftarrow not\ \mathbf{out}(Ann)\\
\mathbf{out}(Bob) &\leftarrow not\ \mathbf{in}(Bob)\\
\mathbf{in}(Chris) &\leftarrow not\ \mathbf{out}(Chris)\\
\mathbf{out}(David) &\leftarrow not\ \mathbf{in}(David)\\
\mathbf{in}(Bob) &\leftarrow \mathbf{out}(Ann)\\
\mathbf{in}(Ann) &\leftarrow \mathbf{out}(Bob)\\
\mathbf{in}(David) &\leftarrow \mathbf{in}(Chris)\\
\mathbf{out}(Chris) &\leftarrow \mathbf{out}(David)\\
\mathbf{out}(Ann) &\leftarrow \mathbf{in}(David)\\
\mathbf{out}(David) &\leftarrow \mathbf{in}(Bob)
\end{aligned}
$$

One can see that $\{\mathbf{in}(Ann), \mathbf{out}(Bob), \mathbf{out}(Chris), \mathbf{out}(David)\}$ is the only coherent stable model of $\mathcal{P}'(P, I)$. It corresponds to $R = \{Ann\}$, which is the only P-justified revision of I as we saw in Example 2.2.

Remark 3.3 *The translation described in this section can be directly applied to revision programs with variables. As a result we get logic programs with variables which can be processed by existing systems for computing stable models (in particular, by a grounder for s-models, lparse [29]).*

3.2 Shifting Theorem

The Przymusinski–Turner embedding of revision programs into logic programs has some drawbacks (Remark 3.1). This section describes a more elegant embedding that does not require an explicit representation of an initial database, and does not increase the size of a program ([18]). This embedding is based on a so called *Shifting Theorem* (Theorem 3.4), which reflects the symmetry between **in**s and **out**s in revision programming.

3.2.1 W-Transformation

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 [23] 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. If α is a literal of the form **in**(a) or **out**(a), then

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

Thus, T_W replaces some literals by their duals and leaves other literals unchanged. Specifically, when $a \in W$, the literals **in**(a) and **out**(a) are replaced by their duals. When $a \notin W$, the literals **in**(a) and **out**(a) remain unchanged.

The definition of T_W naturally extends to sets of literals and sets of atoms. Namely, given a set L of literals, we apply T_W to every literal in the set. In other words,

$$T_W(L) = \{T_W(\alpha)\colon \alpha \in L\}.$$

Given a set of atoms A, we apply T_W to the complete representation of A, $A^c = \{\textbf{in}(a)\colon a \in A\} \cup \{\textbf{out}(a)\colon a \notin A\}$, and write the result as a set of atoms. That is,

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

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

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

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

$$T_I(I) = \emptyset \quad \text{and} \quad T_U(I) = U \setminus I \qquad \text{for any } I \subseteq U.$$

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

Lemma 3.2 *Let S_1 and S_2 be sets of literals or sets of atoms. Then for any $W \subseteq U$,*

1. $T_W(S_1) = T_W(S_2)$ if and only if $S_1 = S_2$;

2. $T_W(S_1 \cup S_2) = T_W(S_1) \cup T_W(S_2)$;

3. $T_W(S_1 \cap S_2) = T_W(S_1) \cap T_W(S_2)$;

4. $T_W(S_1 \setminus S_2) = T_W(S_1) \setminus T_W(S_2)$;

5. $T_W(T_W(S_1)) = S_1$. □

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 $T_W(P) = \{T_W(r) \colon r \in P\}$.

3.2.2 Shifting Theorem

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

Proof.

Let $W = I_1 \div I_2$. When calculating the necessary change, we treat literals as propositional atoms of the form **in**(a) and **out**(b). The W-transformation can be viewed as a renaming of 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,

$$NC(P_{I_1,R_1}) = T_W(NC(T_W(P_{I_1,R_1}))).$$

Let $R_2 = T_W(R_1)$. Observe that by the definition of T_W, the inertia of I_2 and R_2 satisfies the equality $I(I_2, R_2) = T_W(I(I_1, R_1))$. Hence, $T_W(P_{I_1,R_1}) = (T_W(P))_{I_2,R_2}$.

Theorem 2.7 and Lemma 3.2 imply the following sequence of equivalences .

- R_1 is a P-justified revision of I_1,
- $NC(P_{I_1,R_1}) \cup I(I_1, R_1) = R_1^c$,
- $T_W(NC(P_{I_1,R_1}) \cup I(I_1, R_1)) = T_W(R_1^c)$,
- $(T_W(NC(P_{I_1,R_1})) \cup (T_W(I(I_1, R_1))) = T_W(R_1^c)$,
- $NC(T_W(P_{I_1,R_1})) \cup I(I_2, R_2) = T_W(\{\mathbf{in}(a) : a \in R_1\} \cup \{\mathbf{out}(a) : a \notin R_1\})$,
- $NC((T_W(P))_{I_2,R_2}) \cup I(I_2, R_2) = R_2^c$,
- $R_2 = T_W(R_1)$ is a $T_W(P)$-justified revision of I_2.

This proves the statement of the theorem. □

Remark 3.4 *The duality theorem (Theorem 2.12) is a special case of Theorem 3.4 when* $I_2 = U \setminus I_1$.

Corollary 3.1 *For each pair of databases* I *and* R, R *is a* P*-justified revision of* I *if and only if* $T_I(R)$ *is a* $T_I(P)$*-justified revision of* $\emptyset$.

At first glance, a revision problem seems to have two independent parameters: a revision program P that specifies constraints that need to be satisfied, 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 $P' = T_{I \div I'}(P)$. The relation is reflexive, symmetric and transitive. Moreover, by the 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'-justified 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 equivalent revision problems have the same size).

Example 3.3 Let us look at the familiar problem about forming a committee which we considered in Example 2.2. Recall that $I = \{Ann, Chris\}$. Let us apply transformation T_I (shift to the empty initial database). Clearly, $T_I(I) = \emptyset$. Revision program $T_I(P)$ is obtained from P by replacing revision literals **in**(Ann), **out**(Ann), **in**$(Chris)$, **out**$(Chris)$ by **out**(Ann), **in**(Ann), **out**$(Chris)$, **in**$(Chris)$, respectively. It is easy to see that $T_I(P)$ consists of the rules:

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

This revision program has only one justified revision of $\emptyset$, $\{Chris\}$. Observe moreover that $\{Chris\} = T_I(\{Ann\})$, which agrees with the assertion of Theorem 3.4. □

3.2.3 Revision programming = Logic programming + constraints

In this section we describe embedding of revision programs into logic programs based on the Shifting Theorem (Theorem 3.4).

Let us first consider the case of an empty initial database. Every revision program can be divided into two parts: in-rules ($\{r \in P : head(r) = \textbf{in}(a)$ for some $a \in U\}$) and out-rules ($\{r \in P : head(r) = \textbf{out}(a)$ for some $a \in U\}$). Let us denote the set of all in-rules of a revision program P as $InRules(P)$. The set of all out-rules of P is denoted $OutRules(P)$. Clearly, $P = InRules(P) \cup OutRules(P)$. The heads of all rules from $OutRules(P)$ are satisfied in the initial database (which is empty). Hence, they do not describe any changes to the initial database. Any such change must be enforced by $InRules(P)$. The following theorem states that P-justified revisions of the empty set are, in fact, $InRules(P)$-justified revisions of the empty set that satisfy $OutRules(P)$. This is a counterpart of Lemma 2.2 on Horn theories.

Theorem 3.5 *Let P be a revision program. Then R is a P-justified revision of $\emptyset$ if and only if R is an $InRules(P)$-justified revision of $\emptyset$ and R is a model of $OutRules(P)$.*

Proof.

($\Rightarrow$) If R is a P-justified revision of $\emptyset$, then R is a model of P. Hence, it is a model of $OutRules(P)$. It remains to show that R is a $InRules(P)$-justified revision of $\emptyset$. Let $M = NC(P \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\})$. Theorem 2.7 implies that $M = \{in(a) : a \in R\} \cup \{out(a) : a \notin R\}$. By the definition of the necessary change, M is the least model of $P \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\}$ = $InRules(P) \cup OutRules(P) \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\}$. By the definition of inertia, $I(\emptyset, R) = \{out(a) : a \notin R\}$. Let us partition $OutRules(P)$: $OutRules(P) = P_1 \cup P_2$, where $head(P_1) \subseteq \{out(a) : a \in R\}$, $head(P_2) \subseteq \{out(a) : a \notin R\}$. For each rule $r \in P_2$, $head(r) \in I(\emptyset, R)$. Hence, there exists a rule $head(r) \leftarrow$ in the set $\{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\}$. Therefore, M is also the least model of the program $InRules(P) \cup P_1 \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\}$. If we remove from the program some rules whose premises are false in M, M will remain 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 $head(r) \in \{out(a) : a \in R\}$. Assume that premises of r are true in M. Then $head(r)$ must be true in M, since M is the model of $InRules(P) \cup P_1 \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\}$. Hence, $M \cap \{out(a) : a \in R\} \neq \emptyset$. This contradicts the fact that $M = \{in(a) : a \in R\} \cup \{out(a) : a \notin R\}$. Therefore, M is the least model of the program $InRules(P) \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\}$. In other words, $NC(InRules(P) \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\})$ = $\{in(a) : a \in R\} \cup \{out(a) : a \notin R\}$. Theorem 2.7 implies that R is a $InRules(P)$-justified revision of $\emptyset$.

($\Leftarrow$) Assume R is a $InRules(P)$-justified revision of $\emptyset$, and R is a model of $OutRules(P)$. Theorem 2.7 implies that $NC(InRules(P) \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\}) = \{in(a) : a \in R\} \cup \{out(a) : a \notin R\}$. Let $M = \{in(a) : a \in R\} \cup \{out(a) : a \notin R\}$. M is the least model of $InRules(P) \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\}$. Clearly, M is also the least model of a modified program obtained from $(InRules(P) \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\})$ by adding some rules that are satisfied by M. All rules in $OutRules(P)$ are satisfied in M by our assumption. Therefore, M is the least model of

$$InRules(P) \cup OutRules(P) \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\} = P \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\}.$$

Hence, $NC(P \cup \{\alpha \leftarrow\colon \alpha \in I(\emptyset, R)\}) = \{in(a) : a \in R\} \cup \{out(a) : a \notin R\}$. That is, R is a P-justified revision of $\emptyset$. □

Theorem 3.5 implies that out-rules of P may be viewed as additional constraints that $InRules(P)$-justified revisions of the empty set must satisfy in order to be P-justified revisions.

At the same time, $InRules(P)$ may be viewed as a logic program. Indeed, let us define the mapping lp between revision in-rules and logic program clauses. Given a revision 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 logic program clause $c = lp(r)$ as

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

For a set of revision in-rules P we define the corresponding logic program $lp(P)$ to be $\{lp(r) : r \in$P$\}$.

Let us notice that the mapping lp is the inverse of the mapping rp between logic program clauses and revision in-rules (Section 2.4.3).

Theorem 3.5 and Theorem 2.14 imply the following result.

Corollary 3.2 *Let P be a revision program. Then, R is a P-justified revision of $\emptyset$ if and only if R is a stable model of $lp(InRules(P))$ and R is a model of $OutRules(P)$.*

The Shifting Theorem allows us to generalize this result to the case of arbitrary initial databases.

Definition 3.4 *Let I be a database. A revision rule r 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 3.6 *Let P be a revision program. 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 $T_I(R)$ is a stable model of $lp(T_I(P''))$ and R satisfies all rules from P'.*

Proof.
Corollary 3.1 implies that R is a P-justified revision of I if and only if

(3.1) $T_I(R)$ is a $T_I(P)$-justified revision of $\emptyset$.

It is easy to see that $InRules(T_I(P)) = T_I(P'')$, and $OutRules(T_I(P)) = T_I(P')$. Thus, by Corollary 3.2, (3.1) holds if and only if $T_I(R)$ is a stable model of $lp(T_I(P''))$ and a model of $T_I(P')$. Applying T_I-transformation, we get that $T_I(R)$ is a model of $T_I(P')$ if and only if R satisfies all rules in $P' = T_I(T_I(P'))$. Therefore, R is a P-justified revision of I if and only if $T_I(R)$ is a stable model of $lp(T_I(P''))$ and R satisfies all rules in P'. □

Remark 3.5 *Let us note that in the Theorem 3.6, R satisfies all rules from P' if and only if $T_I(R)$ satisfies all rules from $T_I(P')$ (this is implied by the Shifting Theorem). Consequently, the statement of the Theorem 3.6 may be rewritten as follows. A database R is a P-justified revision of I if and only if $T_I(R)$ is a stable model of $lp(T_I(P''))$ that satisfies all constraints from $T_I(P')$.*

Informally, Theorem 3.6 shows that every revision program is equivalent to a logic program and a set of constraints. We can formulate it as a conceptual equality:

revision programming = logic programming + constraints.

Remark 3.6 *Logic programming with constraints was studied before, in particular in [16]. In Section 5.1.2 we show the connection between justified revisions and general logic programs from [16].*

Theorems 3.4, 3.5, and 3.6 define the following embedding of revision programs into logic programs. Given a revision program P and an initial database I, we transform them into the logic program $lp(InRules(T_I(P)))$ and the set of constraints $OutRules(T_I(P))$. Theorem 3.6 implies that there is a one-to one correspondence between P-justified revisions of I and stable models of $lp(InRules(T_I(P)))$ that satisfy $OutRules(T_I(P))$.

The logic program $lp(InRules(T_I(P)))$ is a logic program

1. over the language with $|U|$ propositional letters;
2. that has no more clauses than there are rules in P, and its size is not greater than the size of P (the total number of logic clauses and constraints is equal to the number of rules in P, and the size of the logic program plus the size of constraints is equal to the size of P);
3. such that its stable models do not have superfluous elements to represent the initial database.

Therefore, the embedding based on the Shifting Theorem (Theorem 3.4) does not have the drawbacks of the Przymusinski–Turner embedding outlined in Remark 3.1. It requires, though, performing the T_I-transformation on P and stable models of $lp(InRules(T_I(P)))$ and checking if they satisfy constraints.

Let us illustrate the embedding by an example.

Example 3.4 Consider P and I from Example 2.2. In Example 3.3 we computed that the revision program $T_I(P)$ is

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

Then, the corresponding logic program $lp(InRules(T_I(P)))$ is the following:

$$\begin{aligned}
Bob &\leftarrow Ann\\
David &\leftarrow not\ Chris\\
Chris &\leftarrow not\ David\\
Ann &\leftarrow David
\end{aligned}$$

It has two stable models $M_1 = \{Chris\}$ and $M_2 = \{Ann, Bob, David\}$. However, only M_1 satisfies the constraints ($OutRules(T_I(P))$). After applying the T_I-transformation we get $T_I(M_1) = \{Ann\}$. This is indeed the only P-justified revision of I, as was shown in Example 2.2.

3.2.4 Computing justified revisions

Similarly to Section 3.1.3 we can use Shifting Theorem to embed revision programs into logic programs, compute their stable models and obtain from them justified revisions.

The translation via Shifting Theorem of revision programs into logic programs described in Section 3.2.3 allows us to compute justified revisions according to the following scheme (see also Figure 3.1).

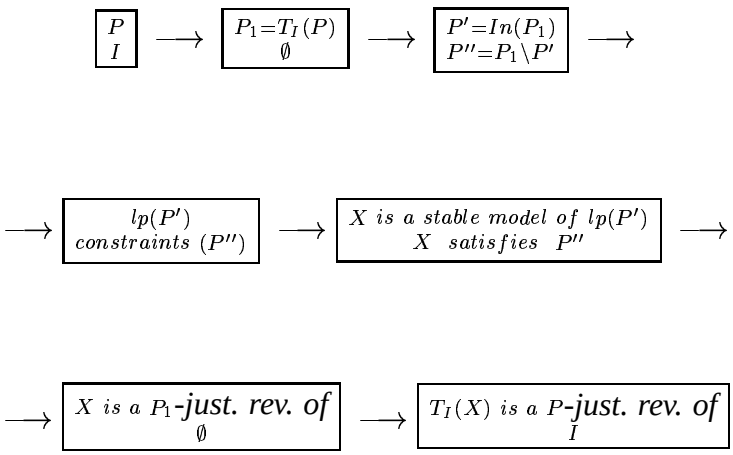

Figure 3.1: Computing justified revisions.

1. Given a revision program P and initial database I, apply transformation T_I to obtain a new revision program $P_1 = T_I(P)$.

2. Let P' be a revision program consisting of in-rules of revision program P_1 ($P' = In(P_1)$). Let P'' be a revision program consisting of out-rules of P_1 ($P'' = P_1 \setminus P'$).

3. Consider the logic program $lp(P')$ which corresponds to P'.

4. Compute stable models of $lp(P')$. They are P'-justified revisions of $\emptyset$.

5. From stable models computed in the previous step, select those which satisfy P''. They are P_1-justified revisions of $\emptyset$.

6. Apply transformation T_I to justified revisions obtained in the previous step to get P-justified revisions of I.

To illustrate the above scheme let us use it to compute justified revisions for the revision program and the initial database from Example 2.2.

Example 3.5 (continuation of Example 2.2) *We need to find P-justified revisions of I, where P and I are from Example 2.2. Using the scheme for computing justified revisions, we get the following.*

1. *Applying transformation T_I to P results in program P_1:*

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

2. *Separating in-rules and out-rules of P_1 results in program P':*

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

   and program P'' (set of constraints):

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

3. *Logic program corresponding to P' is $lp(P')$:*

$$\begin{aligned}
Bob &\leftarrow Ann\\
David &\leftarrow not\ Chris\\
Chris &\leftarrow not\ David\\
Ann &\leftarrow David
\end{aligned}$$

4. *There are two stable models of $lp(P')$: $\{Chris\}$ and $\{Ann, Bob, David\}$. They are also P'-justified revisions of $\emptyset$.*

5. *Database $\{Chris\}$ satisfies constraints from P''. Database $\{Ann, Bob, David\}$ does not satisfy the constraints. Therefore, there is only one P_1-justified revision of $\emptyset$, which is $\{Chris\}$.*

6. *Applying transformation T_I to $\{Chris\}$ results in database $\{Ann\}$. Thus, $\{Ann\}$ is P-justified revision of I.*

Therefore, the desired committee should consist of one person, who is Ann.

Remark 3.7 *The first step of the algorithm described in this section involves shifting a given revision problem to a revision problem with empty initial database. This step can not be performed directly on a revision program with variables. Therefore, revision programs with variables need to be grounded before executing the algorithm.*

Chapter 4

Well-Founded Semantics for Revision Programming

In revision programming a database may admit none, one or many revisions. What if we want to have a single "intended" model?

In the beginning of Section 2.3.2 we mentioned two types of approaches to address the same issue in logic programming. Because of the similarities between revision programming and logic programming the same approaches can be used for revision programming. Marek and Truszczyński in [23] presented a solution of the first type: they restrict class of revision programs to those that have the desired property: to every initial database they assign a unique justified revision.

Definition 4.1 ([23]) *A revision program is* safe *if for every literal* $\alpha \in head(P)$, $\alpha^D \notin var(P)$. $\triangle$

Definition 4.2 ([23]) *A revision program* P *is* stratified *if there exists* $\{P_t\}_{0<t<n}$*, a partition of* P*, such that for every* $0 < t < n$*:*

1. P_t *is safe, and*
2. *if* $\alpha \in head(P_t)$ *then* $\alpha, \alpha^D \notin \cup_{q<t} var(P_q)$. $\triangle$

Safe and stratified revision programs have exactly one justified revision for every initial database.

A solution of the second type is a well-founded semantics for revision programs, where each revision problem (revision program and initial database) is assigned a single 3-valued model on the set of revision literals. In this chapter we discuss different ways of defining well-founded semantics for revision programs.

4.1 Definitions induced by embeddings into logic programming.

Revision programs and logic programs are closely related, as we discussed in Chapter 3. Therefore, given a revision problem, we can translate it into a logic program, find the well-founded semantics for the logic program, and declare the result to be a well-founded model of the revision program. The two ways of embedding of revision programs into logic programs presented in Chapter 3 give rise to two definitions of well-founded semantics for revision programs. In this section we present these two definitions and show that they are indeed different.

4.1.1 A definition obtained via the Przymusinski–Turner translation

In this section we use the Przymusinski–Turner embedding of revision programs into logic programs described in Section 3.1.1 to compute well-founded semantics for revision programs.

Given a revision program P and a database I, we compute the translation of P and I into a logic program, $\mathcal{P}(P, I)$ (as specified in Definition 3.1). Recall that $\mathcal{P}(P, I)$ is a logic program over a propositional language $\mathcal{K}$ with the set of propositional letters $\{\mathbf{in}(q) : q \in U\} \cup \{\mathbf{out}(q) : q \in U\} \cup \{\mathbf{in}_I(q) : q \in U\} \cup \{\mathbf{out}_I(q) : q \in U\}$. To find a well-founded semantics for $\mathcal{P}(P, I)$ we compute a least fixpoint and a greatest fixpoint of the operator $\gamma^2_{\mathcal{P}(P,I)}$. Then, for every revision literal $l \in \{\mathbf{in}(q) : q \in U\} \cup \{\mathbf{out}(q) : q \in U\}$ we can tell whether it is well-founded, unfounded or unknown relative to the logic program $\mathcal{P}(P, I)$. Therefore, we can define a well-founded semantics of revision programs (which we denote WFS^{PT}) as follows.

Definition 4.3 (WFS^{PT}) *Let P be a revision program. Let I be an initial database. The revision literals that belong to the least fixpoint of $\gamma^2_{\mathcal{P}(P,I)}$ are* well-founded^{PT} *relative to P and I. The revision literals that do not belong to the greatest fixpoint of $\gamma^2_{\mathcal{P}(P,I)}$ are* unfounded^{PT} *relative to P and I. The remaining revision literals are called* unknown^{PT}. △

Let us illustrate the definition by an example.

Example 4.1 *Let $I = \emptyset$. Consider*

$$\begin{aligned} P: \quad \mathbf{out}(a) &\leftarrow \mathbf{out}(b) \\ \mathbf{in}(b) &\leftarrow \mathbf{out}(a) \\ \mathbf{in}(a) &\leftarrow \end{aligned}$$

The logic program $\mathcal{P} = \mathcal{P}(P, I)$ is the following.

$$\begin{aligned} \mathcal{P}: \quad \mathbf{out}_I(a) &\leftarrow \\ \mathbf{out}_I(b) &\leftarrow \\ \mathbf{in}(a) &\leftarrow \mathbf{in}_I(a), not\ \mathbf{out}(a) \\ \mathbf{in}(b) &\leftarrow \mathbf{in}_I(b), not\ \mathbf{out}(b) \\ \mathbf{out}(a) &\leftarrow \mathbf{out}_I(a), not\ \mathbf{in}(a) \end{aligned}$$

$$\begin{aligned}
\mathbf{out}(b) &\leftarrow \mathbf{out}_I(b), not\ \mathbf{in}(b) \\
\mathbf{out}(a) &\leftarrow \mathbf{out}(b) \\
\mathbf{in}(b) &\leftarrow \mathbf{out}(a) \\
\mathbf{in}(a) &\leftarrow
\end{aligned}$$

Thus, $lfp(\gamma^2_{\mathcal{P}}) = \{\mathbf{out}_I(a), \mathbf{out}_I(b), \mathbf{in}(a)\}$, $gfp(\gamma^2_{\mathcal{P}}) = \{\mathbf{out}_I(a), \mathbf{out}_I(b), \mathbf{in}(a), \mathbf{out}(a), \mathbf{in}(b), \mathbf{out}(b)\}$. *Therefore, the only well-founded*^{PT} *revision literal is* $\mathbf{in}(a)$. *There are no unfounded*^{PT} *revision literals. Revision literals* $\mathbf{out}(a)$, $\mathbf{in}(b)$, *and* $\mathbf{out}(b)$ *are unknown*^{PT}.

The following result shows that the definition agrees with intuitions. Namely, that well-founded revision literals must be satisfied by all justified revisions, and no justified revision may satisfy an unfounded revision literal.

Theorem 4.1 *Let* P *be a revision program. Let* I *be an initial database. Then, any* P*-justified revision of* I

- *satisfies all revision literals that are well-founded*^{PT} *relative to* P *and* I*, and*
- *satisfies no revision literals unfounded*^{PT} *relative to* P *and* I.

Proof.
Let R be a P-justified revision of I. Let l be a revision literal. By Theorem 3.1, there exists a unique coherent stable model M of $\mathcal{P}(P, I)$ such that $R = \{q : \mathbf{in}(q) \in M\}$.

If l is well-founded^{PT} relative to P and I, then, by definition, it belongs to the least fixpoint of $\gamma^2_{\mathcal{P}(P,I)}$. Hence, l is well-founded relative to $\mathcal{P}(P, I)$. Thus, by Proposition 2.2, $l \in M$. Since M is coherent, this implies that l is satisfied by R.

Similarly, if l is unfounded^{PT} relative to P and I, then it does not belong to the greatest fixpoint of $\gamma^2_{\mathcal{P}(P,I)}$. Hence, l is unfounded relative to $\mathcal{P}(P, I)$. Thus, by Proposition 2.2, $l \notin M$. Consequently, l is not satisfied by R. □

Corollary 4.1 *If for some* $a \in U$ *both* $\mathbf{in}(a)$ *and* $\mathbf{out}(a)$ *are well-founded*^{PT} *relative to* P *and* I*, then there are no* P*-justified revisions of* I.

Proof.

By the theorem, if a justified revision exists it must satisfy both **in**(a) and **out**(a). No database has such a property. □

WFS^{PT} assigns to every revision literal a value (well-founded, unfounded or unknown). We can think of it as assigning to each atom $a \in U$ a pair $\langle \alpha, \beta \rangle$, where $\alpha, \beta \in \{$well-founded, unfounded, unknown$\}$, and α (resp. β) is the value of **in**(a) (resp. **out**(a)). The following theorem shows that not all pairs $\langle \alpha, \beta \rangle$ are valid assignments under WFS^{PT}.

Theorem 4.2 *Let P be a revision program. Let I be a database. Then, for any atom $a \in U$, if a revision literal* **in**(a) *(respectively, a revision literal* **out**(a)*) is unfounded^{PT} then the revision literal* **out**(a) *(respectively,* **in**(a)*) is well-founded^{PT}.*

Proof.

Assume that **in**(a) is unfounded^{PT} relative to P and I. That means that **in**(a) does not belong to the greatest fixpoint of $\gamma^2_{\mathcal{P}(P,I)}$. Let us show that **out**(a) belongs to the least fixpoint of $\gamma^2_{\mathcal{P}(P,I)}$.

Case 1: $a \in I$. Since $\mathbf{in}_I(a) \leftarrow$ is in $\mathcal{P}(P, I)$, $\mathbf{in}_I(a)$ is in all fixpoints of $\gamma^2_{\mathcal{P}(P,I)}$.

The logic program $\mathcal{P}(P, I)$ contains the rule

$$\mathbf{in}(a) \leftarrow \mathbf{in}_I(a), not\ \mathbf{out}(a).$$

Assume that **out**(a) is not in the least fixpoint. Then, **in**(a) must be in the greatest fixpoint. But this contradicts the assumption that **in**(a) is unfounded^{PT} relative to P and I. Consequently, **out**(a) must be in the least fixpoint.

Case 2: $a \notin I$. Then, $\mathbf{out}_I(a)$ is in all fixpoints of $\gamma^2_{\mathcal{P}(P,I)}$.

The logic program $\mathcal{P}(P, I)$ contains the rule

$$\mathbf{out}(a) \leftarrow \mathbf{out}_I(a), not\ \mathbf{in}(a).$$

By our assumption, **in**(a) is not in the greatest fixpoint. Therefore, **out**(a) must be in the least fixpoint.

We showed that if **in**(a) is unfounded^{PT} relative to P and I, then **out**(a) is well-founded^{PT}.

Similarly, we can show that if **out**(a) is unfounded^{PT} relative to P and I, then **in**(a) is well-founded^{PT}. □

Table 4.1: Possible combinations of values for literals **in**(a) and **out**(a) under WFS^{PT}.

in(a)	**out**(a)	revision program P
T	T	$\{\mathbf{in}(a) \leftarrow ; \quad \mathbf{out}(a) \leftarrow \}$
T	U	$\{\mathbf{in}(a) \leftarrow ; \quad \mathbf{out}(a) \leftarrow \mathbf{out}(b) ; \quad \mathbf{in}(b) \leftarrow \mathbf{out}(b) \}$
T	F	$\{\mathbf{in}(a) \leftarrow \}$
U	T	$\{\mathbf{out}(a) \leftarrow ; \quad \mathbf{in}(a) \leftarrow \mathbf{out}(b) ; \quad \mathbf{in}(b) \leftarrow \mathbf{out}(b) \}$
U	U	$\{\mathbf{in}(a) \leftarrow \mathbf{out}(a) \}$
U	F	Impossible by Theorem 4.2
F	T	$\{\}$
F	U	Impossible by Theorem 4.2
F	F	Impossible by Theorem 4.2

Theorem 4.2 implies that there are no more than six pairs of values from {well-founded, unfounded, unknown} which an atom can have under WFS^{PT}. If we denote "well-founded" by T, "unfounded" by F, and "unknown" by U, then these pairs are ⟨T, T⟩, ⟨T, U⟩, ⟨T, F⟩, ⟨U, T⟩, ⟨U, U⟩, ⟨F, T⟩. Every one of these six pairs can be achieved. Table 4.1 provides examples of six revision programs which with an empty initial database give these six different assignments for atom a under WFS^{PT}. Therefore, we can think of WFS^{PT} as a three-valued model on a set of revision literals, or as a six-valued model on a set of atoms.

4.1.2 A definition obtained via Shifting Theorem

Another translation of revision programs into logic programs was described in Section 3.2.3. It is based on the Shifting Theorem. Similarly to the previous section, we take a revision program, translate it into a logic program, find the well-founded semantics of the logic program and declare the corresponding revision literals to be well-founded, unfounded or unknown. The main difference with the Przymusinski–Turner translation is that the Shifting Theorem allows us to translate revision problems into logic programs with constraints. Therefore, in addition to the well-founded semantics of the logic program part of the translation, we also use the constraints and the principle

of inertia to define a well-founded semantics for revision programs.

Let P be a revision program. Let I be an initial database.

Definition 4.4 *For a set of revision literals X define the* reduct *of (P, I) relative to X (denoted by $(P, I)^X$) to be the revision program obtained from P by*

1. *removing every rule $r \in P$ such that $body(r) \cap \{l^D : l \in X \setminus I^c\} \neq \emptyset$,*
2. *deleting from the body of each remaining rule any revision literal that is in I^c.* $\triangle$

Definition 4.5 *For a revision program P and an initial database I, the operator $\gamma_{P,I}$ from sets of revision literals to sets of revision literals is defined by the equation*

$$\gamma_{P,I}(X) = NC((P, I)^X),$$

where $NC((P, I)^X)$ is the necessary change of $(P, I)^X$. $\triangle$

Proposition 4.1 *For any revision program P and initial database I, the operator $\gamma_{P,I}$ is anti-monotone.*

Proof.
Let X_1 and X_2 be sets of revision literals, $X_1 \subseteq X_2$. Then, $X_1 \setminus I^c \subseteq X_2 \setminus I^c$. Hence, $\{l^D : l \in X_1 \setminus I^c\} \subseteq \{l^D : l \in X_2 \setminus I^c\}$. Therefore, $(P, I)^{X_2} \subseteq (P, I)^{X_1}$. Consequently, $\gamma_{P,I}(X_2) \subseteq \gamma_{P,I}(X_1)$. □

Since $\gamma_{P,I}$ is anti-monotone, the operator $\gamma^2_{P,I}$ is monotone. By the Knaster-Tarski Theorem, $\gamma^2_{P,I}$ has a least fixpoint and a greatest fixpoint.

Definition 4.6 (WFS^{Sh}) *Let P be a revision program. Let I be a database. The revision literals that belong to the $lfp(\gamma^2_{P,I}) \cup \{l \in I^c : l^D \notin gfp(\gamma^2_{P,I})\}$ are* well-founded^{Sh} *relative to (P, I). The revision literals that do not belong to the $gfp(\gamma^2_{P,I}) \cup I^c$ are* unfounded^{Sh} *relative to (P, I). The remaining revision literals are* unknown^{Sh} *relative to (P, I).* $\triangle$

The following example illustrates the definition.

Example 4.2 *Let $I = \emptyset$. Consider*

$$\begin{array}{rrcl} P: & \textbf{out}(a) & \leftarrow & \\ & \textbf{out}(b) & \leftarrow & \\ & \textbf{in}(b) & \leftarrow & \textbf{out}(a) \\ & \textbf{in}(a) & \leftarrow & \textbf{out}(b) \end{array}$$

Then, $lfp(\gamma^2_{P,I}) = \{\textbf{out}(a), \textbf{out}(b)\}$, and $gft(\gamma^2_{P,I}) = \{\textbf{out}(a), \textbf{out}(b), \textbf{in}(a), \textbf{in}(b)\}$. Therefore, revision literals $\textbf{out}(a)$, $\textbf{out}(b)$ are well-founded^{Sh}, no revision literals are unfounded^{Sh}, revision literals $\textbf{in}(a)$, $\textbf{in}(b)$ are unknown^{Sh}.

Theorem 4.3 *Let Π be a logic program. Let W be a set of well-founded atoms relative to Π. Let N be a set of unfounded atoms relative to Π. Then, $\{\textbf{in}(a) : a \in W\} \cup \{\textbf{out}(a) : a \in N\}$ is the set of well-founded^{Sh} revision literals relative to $(rp(\Pi), \emptyset)$, and $\{\textbf{in}(a) : a \in N\}$ is the set of unfounded^{Sh} revision literals.*

Proof.

Since $rp(\Pi)$ has only revision literals of the form $\textbf{in}(a)$ ($a \in U$) in the heads of the rules, $\gamma_{rp(\Pi),\emptyset}(X) \subseteq \{\textbf{in}(a) : a \in U\}$ for any set of revision literals X. Therefore, all revision literals of the form $\textbf{out}(a)$ ($a \in U$) are unfounded relative to $(rp(\Pi), \emptyset)$.

For a rule $r \in rp(\Pi)$, $body(r) \cap \{l^D : l \in X \setminus \emptyset^c\} \neq \emptyset$ if and only if the body of r contains a revision literal $\textbf{out}(b)$ for some $\textbf{in}(b) \in X$. This is the case only when the logic program clause $lp(r)$ has the literal $not\ b$ in the body for $b \in X+$, where $X+ = \{a \in U : \textbf{in}(a) \in X\}$. Therefore, the first step in the definition of the reduct $(rp(\Pi), \emptyset)^X$ corresponds to the first step of the definition of the reduct Π^{X+}. That is, $r \in rp(\Pi)$ is deleted on the first step of computing $(rp(\Pi), \emptyset)^X$ if and only if the corresponding logic program clause $lp(r) \in \Pi$ is deleted on the first step of computing Π^{X+}.

In the second step of the definition of $(rp(\Pi), \emptyset)^X$ all revision literals of the form $\textbf{out}(a)$ ($a \in U$) are deleted from the remaining rules. It corresponds to deleting all atoms with negation as failure from the bodies of the remaining logic program clauses. Consequently, $rp(\Pi^{X+}) = (rp(\Pi), \emptyset)^X$.

Since the revision program $(rp(\Pi), \emptyset)^X$ has only revision literals of the form $\mathbf{in}(a)$ ($a \in U$), $\gamma_\Pi(X+) = \gamma_{rp(\Pi),\emptyset}(X)$ for any set of revision literals X. Therefore, $Y \subseteq U$ is a fixpoint of γ^2_Π if and only if $\{\mathbf{in}(a) : a \in Y\}$ is a fixpoint of $\gamma^2_{rp(\Pi),\emptyset}$.

By definition, well-founded^{Sh} revision literals relative to $(rp(\Pi), \emptyset)$ are

$$lfp(\gamma^2_{rp(\Pi),\emptyset}) \cup \{l \in \emptyset^c : l^D \notin gfp(\gamma^2_{rp(\Pi),\emptyset})\} =$$

$$\{\mathbf{in}(a) : a \in lfp(\gamma^2_\Pi)\} \cup \{\mathbf{out}(a) : \mathbf{in}(a) \notin gfp(\gamma^2_{rp(\Pi),\emptyset})\} =$$

$$\{\mathbf{in}(a) : a \in W\} \cup \{\mathbf{out}(a) : a \notin gfp(\gamma^2_\Pi)\} =$$

$$\{\mathbf{in}(a) : a \in W\} \cup \{\mathbf{out}(a) : a \in N\}.$$

By definition, unfounded^{Sh} revision literals relative to $(rp(\Pi), \emptyset)$ are revision literals that do not belong to the $gfp(\gamma^2_{rp(\Pi),\emptyset}) \cup I^c$. Hence, $\{\mathbf{in}(a) : \mathbf{in}(a) \notin gfp(\gamma^2_{rp(\Pi),\emptyset})\}$ is the set of unfounded^{Sh} revision literals. However,

$$\{\mathbf{in}(a) : \mathbf{in}(a) \notin gfp(\gamma^2_{rp(\Pi),\emptyset})\} =$$

$$\{\mathbf{in}(a) : a \notin gfp(\gamma^2_\Pi)\} = \{\mathbf{in}(a) : a \in N\}.$$

This finishes the proof. □

Theorem 4.4 *Let P be a revision program. Let I be an initial database. Then, any P-justified revision of I*

- *satisfies all revision literals that are well-founded^{Sh} relative to P and I, and*
- *satisfies no revision literals unfounded^{Sh} relative to P and I.*

Proof.
By definition of P-justified revision, $R = I \oplus NC(P_R|I)$. Let X be a set of heads of P_R. By Theorem 2.6, $X = NC(P_R|I)$. Thus, $R = I \oplus X$.

Let us show that X is a fixpoint of $\gamma^2_{P,I}$. To do this we compare the reducts $(P, I)^X$ and $P_R|I$. Recall, that $P_R|I$ (Definition 2.16) is obtained from P by

1. removing every rule $r \in P$ whose body is not satisfied by R;
2. deleting from the body of each remaining rule any revision literal that is in I^c.

Assume that rule $r \in P$ is removed during the first step of computation of the reduct $(P, I)^X$. By definition of the reduct, there exists revision literal l such that $l^D \in body(r)$, $l \in X$, and $l \notin I^c$. From $R = I \oplus X$ and $l \in X$ it follows that $R \models l$. Hence, $R \not\models l^D$. Consequently, the body of rule r is not satisfied by R. Hence, r is removed during the first step of computation of the reduct $P_R|I$. The second steps of computation of the reducts $(P, I)^X$ and $P_R|I$ are the same. Therefore, $P_R|I \subseteq (P, I)^X$.

Assume that rule $r \in P$ is removed during the first step of computation of the reduct $P_R|I$. It means that there exists literal l in the body of r such that $R \not\models l$. Thus, $R \models l^D$. Since $R = I \oplus X$, only the following three cases are possible.

1. $l^D \in X$ and $l^D \in I^c$. Since X is coherent, $l \notin X$. If there are no other literals that cause r to be deleted during the first step of computation of $(P, I)^X$, then the reduct $(P, I)^X$ contains rule r' which is obtained from r by deleting from the body all revision literals that are in I^c. Note, that r' contains l in the body.

2. $l^D \in X$ and $l^D \notin I^c$. Hence, $l^D \in X \setminus I^c$, and $l = (l^D)^D$ is in the body of r. Thus, rule r is deleted during the first step of computation of the reduct $(P, I)^X$.

3. $l^D \notin X$, $l^D \in I^c$, and $l \notin X$. Similarly to case 1, if r is *not* deleted during the first step of computation of $(P, I)^X$, then the reduct $(P, I)^X$ contains rule r' which is obtained from r by deleting from the body all revision literals that are in I^c. Note, that r' contains l in the body.

From the above observations it follows that $(P, I)^X = (P_R|I) \cup S$, where S is a set of rules (possibly empty) with the property that every rule $r \in S$ contains in its body a literal l such that $l \notin X$. We have that $X = NC(P_R|I)$. Since all rules in S have literals that are *not* in X in their bodies, $NC((P, I)^X) = NC((P_R|I) \cup S) = NC(P_R|I) = X$. Thus, $\gamma_{P,I}(X) = NC((P, I)^X) = X$. Hence, X is a fixpoint of $\gamma^2_{P,I}$. By Proposition 2.1, $lfp(\gamma^2_{P,I}) \subseteq X \subseteq gfp(\gamma^2_{P,I})$.

Let l be a well-founded^{Sh} revision literal relative to P and I. By definition, $l \in lfp(\gamma^2_{P,I}) \cup \{l \in I^c : l^D \notin gfp(\gamma^2_{P,I})\}$. There are two possible cases.

Case 1: $l \in lfp(\gamma^2_{P,I})$. Then, $l \in X$. Hence, $R = I \oplus X$ implies $R \models l$.

Case 2: $l \in I^c$ and $l^D \notin gfp(\gamma^2_{P,I})$. Then, $l^D \notin X$. Hence, $R = I \oplus X$ implies $R \models l$.

Therefore, R satisfies all well-founded^{Sh} revision literals.

Let l be an unfounded^{Sh} revision literal relative to P and I. By definition, $l \notin gfp(\gamma^2_{P,I}) \cup I^c$. Thus, $l \notin I^c$ and $l \notin X$. In other words, $l^D \in I^c$ and $(l^D)^D \notin X$. This implies that $R \models l^D$. Hence, $R \not\models l$. Consequently, R satisfies no unfounded^{Sh} revision literals. □

4.1.3 Comparison of the two definitions

In this section we compare the two definitions of well-founded semantics for revision programs from Section 4.1.1 and Section 4.1.2.

We say that one well-founded semantics is *better* (or *more informative*) than the other one on a particular instance of a revision problem if the union of well-founded and unfounded revision literals in the first semantics contains the union of well-founded and unfounded revision literals in the second semantics.

We present examples that show that these definitions are indeed different and neither of them is always better than the other. In particular, WFS^{Sh} is more informative than WFS^{PT} in Example 4.4, whereas WFS^{PT} is better than WFS^{Sh} in Example 4.3.

The following example shows that WFS^{PT} may be more informative than WFS^{Sh}.

Example 4.3 *Let $I = \emptyset$. Consider*

$$\begin{aligned} P: \quad \mathbf{in}(a) &\leftarrow \mathbf{out}(b) \\ \mathbf{in}(b) &\leftarrow \mathbf{out}(a) \\ \mathbf{out}(b) &\leftarrow \end{aligned}$$

1. *Compute WFS^{Sh}. We have: $lfp(\gamma^2_{P,I}) = \{\mathbf{out}(b)\}$, and $gft(\gamma^2_{P,I}) = \{\mathbf{in}(a), \mathbf{in}(b), \mathbf{out}(b)\}$. Hence, well-founded^{Sh} revision literals are $\{\mathbf{out}(b)\}$. There are no unfounded^{Sh} revision literals.*

2. *Compute WFS^{PT}. The logic program $\mathcal{P} = \mathcal{P}(P, I)$ is the following.*

$$\begin{aligned} \mathcal{P}: \quad \mathbf{out}_I(a) &\leftarrow \\ \mathbf{out}_I(b) &\leftarrow \\ \mathbf{in}(a) &\leftarrow \mathbf{in}_I(a), not\ \mathbf{out}(a) \end{aligned}$$

$$\begin{aligned}
\mathbf{out}(a) &\leftarrow \mathbf{out}_I(a), not\ \mathbf{in}(a)\\
\mathbf{in}(b) &\leftarrow \mathbf{in}_I(b), not\ \mathbf{out}(b)\\
\mathbf{out}(b) &\leftarrow \mathbf{out}_I(b), not\ \mathbf{in}(b)\\
\mathbf{in}(a) &\leftarrow \mathbf{out}(b)\\
\mathbf{in}(b) &\leftarrow \mathbf{out}(a)\\
\mathbf{out}(b) &\leftarrow
\end{aligned}$$

Thus, $lfp(\gamma_{\mathcal{P}}^2) = gfp(\gamma_{\mathcal{P}}^2) = \{\mathbf{out}_I(a), \mathbf{out}_I(b), \mathbf{in}(a), \mathbf{out}(b)\}$. *Therefore, revision literals* $\mathbf{in}(a)$, $\mathbf{out}(b)$ *are well-founded*^{PT}. *Revision literals* $\mathbf{out}(a)$, $\mathbf{in}(b)$ *are unfounded*^{PT}.

Consequently, WFS^{PT} *is more informative than WFS*^{Sh} *relative to* P *and* I.

It is easy to see that $R = \{a\}$ *is a* P*-justified revision of* I. *WFS*^{PT} *allows us to conclude that* a *must be in possible revisions, and* b *must be out. Whereas WFS*^{Sh} *only implies that* P*-justified revisions of* $\emptyset$ *must satisfy* $\mathbf{out}(b)$.

The following example shows that WFS^{Sh} may be more informative than WFS^{PT}.

Example 4.4 *Let* $I = \emptyset$. *Consider*

$$\begin{aligned}
P:\ \mathbf{in}(a) &\leftarrow \mathbf{out}(b)\\
\mathbf{in}(b) &\leftarrow \mathbf{out}(a)\\
\mathbf{in}(c) &\leftarrow \mathbf{out}(d)\\
\mathbf{in}(d) &\leftarrow \mathbf{out}(c), \mathbf{out}(f)\\
\mathbf{out}(f) &\leftarrow \mathbf{in}(a), \mathbf{in}(b)\\
\mathbf{in}(f) &\leftarrow\\
\mathbf{in}(a) &\leftarrow \mathbf{in}(c), \mathbf{in}(f)
\end{aligned}$$

1. *Compute WFS^{Sh}. We start with $\emptyset$ and obtain iterations of $\gamma_{P,\emptyset}$:*

$$\emptyset \mapsto \left\{ \begin{array}{c} \mathbf{in}(a) \\ \mathbf{in}(b) \\ \mathbf{in}(c) \\ \mathbf{in}(d) \\ \mathbf{out}(f) \\ \mathbf{in}(f) \end{array} \right\} \mapsto \{\mathbf{in}(f)\} \mapsto \left\{ \begin{array}{c} \mathbf{in}(a) \\ \mathbf{in}(b) \\ \mathbf{in}(c) \\ \mathbf{out}(f) \\ \mathbf{in}(f) \end{array} \right\} \mapsto \left\{ \begin{array}{c} \mathbf{in}(c) \\ \mathbf{in}(f) \\ \mathbf{in}(a) \end{array} \right\} \mapsto \left\{ \begin{array}{c} \mathbf{in}(a) \\ \mathbf{in}(c) \\ \mathbf{in}(f) \end{array} \right\}.$$

   Thus, $lfp(\gamma^2_{P,I}) = gft(\gamma^2_{P,I}) = \{\mathbf{in}(a), \mathbf{in}(c), \mathbf{in}(f)\}$. Hence, $\mathbf{in}(a)$, $\mathbf{in}(c)$, $\mathbf{in}(f)$, $\mathbf{out}(b)$, $\mathbf{out}(d)$ are well-founded^{Sh}. Revision literals $\mathbf{in}(b)$, $\mathbf{in}(d)$ are unfounded^{Sh}.

2. *Compute WFS^{PT}. The logic program $\mathcal{P} = \mathcal{P}(P, \emptyset)$ is the following.*

$$\begin{array}{rrl}
\mathcal{P}: & \mathbf{out}_I(a) & \leftarrow \\
& \mathbf{out}_I(b) & \leftarrow \\
& \mathbf{out}_I(c) & \leftarrow \\
& \mathbf{out}_I(d) & \leftarrow \\
& \mathbf{out}_I(f) & \leftarrow \\
& \mathbf{in}(a) & \leftarrow \mathbf{in}_I(a), not\ \mathbf{out}(a) \\
& \mathbf{out}(a) & \leftarrow \mathbf{out}_I(a), not\ \mathbf{in}(a) \\
& \mathbf{in}(b) & \leftarrow \mathbf{in}_I(b), not\ \mathbf{out}(b) \\
& \mathbf{out}(b) & \leftarrow \mathbf{out}_I(b), not\ \mathbf{in}(b) \\
& \mathbf{in}(c) & \leftarrow \mathbf{in}_I(c), not\ \mathbf{out}(c) \\
& \mathbf{out}(c) & \leftarrow \mathbf{out}_I(c), not\ \mathbf{in}(c) \\
& \mathbf{in}(d) & \leftarrow \mathbf{in}_I(d), not\ \mathbf{out}(d) \\
& \mathbf{out}(d) & \leftarrow \mathbf{out}_I(d), not\ \mathbf{in}(d) \\
& \mathbf{in}(f) & \leftarrow \mathbf{in}_I(f), not\ \mathbf{out}(f) \\
& \mathbf{out}(f) & \leftarrow \mathbf{out}_I(f), not\ \mathbf{in}(f) \\
& \mathbf{in}(a) & \leftarrow \mathbf{out}(b) \\
& \mathbf{in}(b) & \leftarrow \mathbf{out}(a)
\end{array}$$

$$\begin{aligned}
\mathbf{in}(c) &\leftarrow \mathbf{out}(d)\\
\mathbf{in}(d) &\leftarrow \mathbf{out}(c), \mathbf{out}(f)\\
\mathbf{out}(f) &\leftarrow \mathbf{in}(a), \mathbf{in}(b)\\
\mathbf{in}(f) &\leftarrow\\
\mathbf{in}(a) &\leftarrow \mathbf{in}(c), \mathbf{in}(f)
\end{aligned}$$

We have the following computation (iterations of $\gamma_{\mathcal{P}}$ starting from $\emptyset$):

$$\emptyset \mapsto \left\{ \begin{array}{c} \mathbf{out}_I(a), \mathbf{out}_I(b), \mathbf{out}_I(c), \mathbf{out}_I(d), \mathbf{out}_I(f), \mathbf{out}(a), \mathbf{out}(b),\\ \mathbf{out}(c), \mathbf{out}(d), \mathbf{out}(f), \mathbf{in}(a), \mathbf{in}(b), \mathbf{in}(c), \mathbf{in}(d), \mathbf{in}(f) \end{array} \right\} \mapsto$$

$$\mapsto \left\{ \begin{array}{c} \mathbf{out}_I(a), \mathbf{out}_I(b),\\ \mathbf{out}_I(c), \mathbf{out}_I(d),\\ \mathbf{out}_I(f), \mathbf{in}(f) \end{array} \right\} \mapsto \left\{ \begin{array}{c} \mathbf{out}_I(a), \mathbf{out}_I(b), \mathbf{out}_I(c), \mathbf{out}_I(d), \mathbf{out}_I(f),\\ \mathbf{out}(a), \mathbf{out}(b), \mathbf{out}(c), \mathbf{out}(d), \mathbf{in}(a),\\ \mathbf{in}(b), \mathbf{in}(c), \mathbf{out}(f), \mathbf{in}(f), \mathbf{in}(d) \end{array} \right\}.$$

Thus, $lfp(\gamma^2_{\mathcal{P}}) = \{\mathbf{in}(f)\} \cup \{\mathbf{out}_I(x) : x \in \{a, b, c, d, f\}\}$ and $gft(\gamma^2_{\mathcal{P}}) = \{\mathbf{out}(a), \mathbf{out}(b), \mathbf{out}(c), \mathbf{out}(d), \mathbf{in}(a), \mathbf{in}(b), \mathbf{in}(c), \mathbf{out}(f), \mathbf{in}(f), \mathbf{in}(d)\} \cup \{\mathbf{out}_I(x) : x \in \{a, b, c, d, f\}\}$. Therefore, the only well-founded^{PT} revision literal is $\mathbf{in}(f)$. There are no unfounded^{PT} revision literals.

Consequently, WFS^{Sh} is more informative than WFS^{PT} relative to P and I.

Moreover, WFS^{Sh} allows us to conclude that the only P-justified revision of $\emptyset$ is $R = \{a, c, f\}$. Whereas WFS^{PT} only imply that P-justified revisions of $\emptyset$ must satisfy $\mathbf{in}(f)$.

4.2 Definition specific to revision programming

In this section we give a definition of well-founded semantics which is specific for revision programming.

4.2.1 Well-founded semantics for revision programming

Definition 4.7 *Given a revision program P and a set of literals A, the revision program $Simpl(P, A)$ is obtained from P by*

1. *removing all rules $r \in P$ such that $body(r) \cap \{l^D : l \in A\} \neq \emptyset$;*

2. *remove all rules with heads from A;*

3. *remove from the bodies of the remaining rules all revision literals that are in A.* $\triangle$

Lemma 4.1 *Let P_1, P_2 be revision programs. Let A_1, A_2 be sets of revision literals. Then,*

1. $Simpl(P_1 \cup P_2, A_1) = Simpl(P_1, A_1) \cup Simpl(P_2, A_2)$, *and*

2. $Simpl(P_1, A_1 \cup A_2) = Simpl(Simpl(P_1, A_1), A_2)$.

Definition 4.8 *Let us define a sequence of triples (P_k, A_k, X_k), $k = 0, 1, \ldots$, as follows.*

- $k = 0$. *Let $P_0 = P$, $A_0 = \emptyset$, $X_0 = \emptyset$.*

- $k \geq 1$. *If $k = 2i + 1$ ($i \in \mathbb{N}$), then $A'_{2i+1} = NC(P_{2i})$.*
  Else if $k = 2i$ ($i \in \mathbb{N}$), then $A'_{2i} = \{l \in I^c : (l^D \notin X_{2i-1})\ \&\ (l \notin A_{2i-1})\ \&\ (l^D \notin A_{2i-1})\}$.
  Let $A_k = A_{k-1} \cup A'_k$.
  If A_k is incoherent then for all $k' \geq k$ let $A_{k'} = A_k$, $X_{k'} = \emptyset$, and $P_{k'} = P_{k-1}$.
  Otherwise, let $P_k = Simpl(P_{k-1}, A'_k)$, $X_k = \gamma_{P_k,I}(X_{k-1})$. $\triangle$

To illustrate the notion, let us compute a sequence of triples for the following example.

Example 4.5 *Let $I = \emptyset$. Consider*

$$
\begin{array}{rrl}
P: & \mathbf{out}(a) & \leftarrow \\
 & \mathbf{in}(a) & \leftarrow \ \mathbf{out}(b) \\
 & \mathbf{in}(b) & \leftarrow \ \mathbf{out}(a) \\
 & \mathbf{in}(c) & \leftarrow \ \mathbf{in}(a), \mathbf{in}(b) \\
 & \mathbf{out}(d) & \leftarrow \ \mathbf{out}(c) \\
 & \mathbf{in}(d) & \leftarrow \ \mathbf{out}(e) \\
 & \mathbf{in}(e) & \leftarrow \ \mathbf{out}(d) \\
 & \mathbf{in}(f) & \leftarrow \ \mathbf{in}(d), \mathbf{in}(e) \\
 & \mathbf{out}(g) & \leftarrow \ \mathbf{out}(f) \\
 & \mathbf{in}(g) & \leftarrow \ \mathbf{out}(h) \\
 & \mathbf{in}(h) & \leftarrow \ \mathbf{out}(g) \\
 & \mathbf{in}(i) & \leftarrow \ \mathbf{in}(g), \mathbf{in}(h)
\end{array}
$$

We have the following computation:

$$A_0 = \emptyset, X_0 = \emptyset, P_0 = P \mapsto$$

$$A_1 = \left\{ \begin{array}{c} \mathbf{out}(a) \\ \mathbf{in}(b) \end{array} \right\}, P_1 = \left\{ \begin{array}{rcl} \mathbf{out}(d) & \leftarrow & \mathbf{out}(c) \\ \mathbf{in}(d) & \leftarrow & \mathbf{out}(e) \\ \mathbf{in}(e) & \leftarrow & \mathbf{out}(d) \\ \mathbf{in}(f) & \leftarrow & \mathbf{in}(d), \mathbf{in}(e) \\ \mathbf{out}(g) & \leftarrow & \mathbf{out}(f) \\ \mathbf{in}(g) & \leftarrow & \mathbf{out}(h) \\ \mathbf{in}(h) & \leftarrow & \mathbf{out}(g) \\ \mathbf{in}(i) & \leftarrow & \mathbf{in}(g), \mathbf{in}(h) \end{array} \right\}, X_1 = \left\{ \begin{array}{c} \mathbf{out}(d), \\ \mathbf{in}(d), \\ \mathbf{in}(e), \\ \mathbf{in}(f), \\ \mathbf{out}(g), \\ \mathbf{in}(g), \\ \mathbf{in}(h), \\ \mathbf{in}(i) \end{array} \right\} \mapsto$$

$$\mapsto A_2 = \left\{ \begin{array}{c} \mathbf{out}(a) \\ \mathbf{in}(b) \\ \mathbf{out}(c) \end{array} \right\}, P_2 = \left\{ \begin{array}{rcl} \mathbf{out}(d) & \leftarrow & \\ \mathbf{in}(d) & \leftarrow & \mathbf{out}(e) \\ \mathbf{in}(e) & \leftarrow & \mathbf{out}(d) \\ \mathbf{in}(f) & \leftarrow & \mathbf{in}(d), \mathbf{in}(e) \\ \mathbf{out}(g) & \leftarrow & \mathbf{out}(f) \\ \mathbf{in}(g) & \leftarrow & \mathbf{out}(h) \\ \mathbf{in}(h) & \leftarrow & \mathbf{out}(g) \\ \mathbf{in}(i) & \leftarrow & \mathbf{in}(g), \mathbf{in}(h) \end{array} \right\}, X_2 = \left\{ \begin{array}{c} \mathbf{out}(d) \end{array} \right\} \mapsto$$

$$\mapsto A_3 = \left\{ \begin{array}{c} \mathbf{out}(a), \\ \mathbf{in}(b), \\ \mathbf{out}(c), \\ \mathbf{out}(d), \\ \mathbf{in}(e) \end{array} \right\}, P_3 = \left\{ \begin{array}{rcl} \mathbf{out}(g) & \leftarrow & \mathbf{out}(f) \\ \mathbf{in}(g) & \leftarrow & \mathbf{out}(h) \\ \mathbf{in}(h) & \leftarrow & \mathbf{out}(g) \\ \mathbf{in}(i) & \leftarrow & \mathbf{in}(g), \mathbf{in}(h) \end{array} \right\}, X_3 = \left\{ \begin{array}{c} \mathbf{out}(g) \\ \mathbf{in}(g) \\ \mathbf{in}(h) \\ \mathbf{in}(i) \end{array} \right\} \mapsto$$

$$\mapsto A_4 = \left\{ \begin{array}{c} \mathbf{out}(a), \\ \mathbf{in}(b), \\ \mathbf{out}(c), \\ \mathbf{out}(d), \\ \mathbf{in}(e), \\ \mathbf{out}(f) \end{array} \right\}, P_4 = \left\{ \begin{array}{rcl} \mathbf{out}(g) & \leftarrow & \\ \mathbf{in}(g) & \leftarrow & \mathbf{out}(h) \\ \mathbf{in}(h) & \leftarrow & \mathbf{out}(g) \\ \mathbf{in}(i) & \leftarrow & \mathbf{in}(g), \mathbf{in}(h) \end{array} \right\}, X_4 = \left\{ \begin{array}{c} \mathbf{out}(g) \end{array} \right\} \mapsto$$

$$\mapsto A_5 = \left\{ \ \mathbf{out}(a), \mathbf{in}(b), \mathbf{out}(c), \mathbf{out}(d), \mathbf{in}(e), \mathbf{out}(f), \mathbf{out}(g), \mathbf{in}(h) \ \right\}, P_5 = \emptyset, X_5 = \emptyset.$$

For $i \geq 6$, $A_i = A_5$, $P_i = \emptyset$, and $X_i = \emptyset$.

Lemma 4.2 *Let P be a revision program. Let I be an initial database. Let R be a P-justified revision of I. Then, $R^c = \gamma_{P,I}(R^c) \cup I(I, R)$.*

Proof.

By Theorem 2.7, $R^c = NC(P_{I,R}) \cup I(I, R)$. By definition, $\gamma_{P,I}(R^c) = NC((P, I)^{R^c})$. Let us show that $NC((P, I)^{R^c}) = NC(P_{I,R})$.

Indeed, the reduct $P_{I,R}$ is obtained from P by removing from the bodies of rules literals from $I(I, R)$. Consider the second step in the definition of the reduct $(P, I)^{R^c}$. Assume that a revision literal l is removed from the body of a rule r. Then, $l \in I^c$. If $l \notin R^c$, then $l^D \in R^c \setminus I^c$, and rule r would have been removed at the first step of the definition of the reduct $(P, I)^{R^c}$. Hence, $l \in R^c$, and thus, $l \in I(I, R)$. Therefore, at the second step of the definition of the reduct $(P, I)^{R^c}$, literals from $I(I, R)$ are removed from the bodies of the remaining rules. Thus, $(P, I)^{R^c} \subseteq P_{I,R}$.

Bodies of the rules which are removed during the first step of the definition of the reduct $(P, I)^{R^c}$ are *not* satisfied by R^c. Since $NC(P_{I,R}) \subseteq R^c$, they are not satisfied by $NC(P_{I,R})$, too. The necessary change of a program remains the same if we remove from the program rules that are not satisfied by the necessary change. Consequently, $NC((P, I)^{R^c}) = NC(P_{I,R})$. Therefore, $R^c = \gamma_{P,I}(R^c) \cup I(I, R)$. □

Lemma 4.3 *Let P be a revision program. Let I be an initial database. Let R be a P-justified revision of I. Let A be a set of literals such that $A \subseteq R^c$. Let $P' = Simpl(P, A) \cup \{l \leftarrow : l \in A\}$. Then, R is a P'-justified revision of I.*

Proof.

By Theorem 2.7,

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

By definition of the necessary change, R^c is the least model of $P \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}$ when treated as a Horn program built of independent propositional atoms of the form $\mathbf{in}(a)$ and $\mathbf{out}(b)$.

Since $A \subseteq R^c$, rules $\{l \leftarrow : l \in A\}$ are satisfied in R^c. Therefore by Lemma 2.1,

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

Let $Q = P \cup \{l \leftarrow : l \in A\} \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}$. We have $NC(Q) = R^c$. It is easy to see that applying steps 1, 2, and 3 from the definition of $Simpl(P, A)$ to the part P of the program Q does not change the least model of the program. In other words,

$$NC(Q) = NC(Simpl(P, A) \cup \{l \leftarrow : l \in A\} \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}).$$

Thus, $R^c = NC(P' \cup \{\alpha \leftarrow : \alpha \in I(I, R)\})$. Therefore, by Theorem 2.7, R is a P'-justified revision of I. □

Lemma 4.4 *Let P be a revision program. Let I be an initial database. Let $A = NC(P)$. Let $P' = Simpl(P, A) \cup \{l \leftarrow : l \in A\}$. Then, R is a P-justified revision of I if and only if R is a P'-justified revision of I.*

Proof.
($\Longrightarrow$) Let R be a P-justified revision of I. Then, by Theorem 2.7,

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

Thus, $A \subseteq R^c$. By Lemma 4.3, R is a P'-justified revision of I.
($\Longleftarrow$) Let R be a P'-justified revision of I. Then,

$$NC(Simpl(P, A) \cup \{l \leftarrow : l \in A\} \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}) = R^c.$$

Notice that we can "undo" steps 3, 2, 1 in the definition of $Simpl(P, A)$ without changing the necessary change of the program

$$(Simpl(P, A) \cup \{l \leftarrow : l \in A\} \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}).$$

Thus,

$$R^c = NC(P \cup \{l \leftarrow : l \in A\} \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}).$$

Since $A = NC(P)$, we get that

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

Consequently, $R^c = NC(P \cup \{\alpha \leftarrow : \alpha \in I(I, R)\})$. Therefore, by Theorem 2.7, R is a P-justified revision of I. □

Lemma 4.5 *Let Q be a revision program. Let I be an initial database. Let X be a set of revision literals such that for every R which is a Q-justified revision of I, $R^c \subseteq X \cup I^c$. Let $A = \{l \in I^c : l^D \notin X\}$. Let $Q' = Simpl(Q, A) \cup \{l \leftarrow : l \in A\}$. Then, R is a Q-justified revision of I if and only if R is a Q'-justified revision of I.*

Proof.
($\Longrightarrow$) Let R be a Q-justified revision of I. Then, $R^c \subseteq X \cup I^c$. Let us show that $A \subseteq R^c$. Let $l \in A$. Then, $l \in I^c$ and $l^D \notin X$. Since I^c does not contain dual revision literals, $l \in I^c$ implies $l^D \notin I^c$. Hence, $l^D \notin X \cup I^c$. Thus, $l^D \notin R^c$. That is, $l \in R^c$. Therefore, $A \subseteq R^c$. By Lemma 4.3, R is a Q'-justified revision of I.
($\Longleftarrow$) Let R be a Q'-justified revision of I. Then,

$$NC(Simpl(Q, A) \cup \{l \leftarrow : l \in A\} \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}) = R^c.$$

Notice that we can "undo" steps 3, 2, 1 in the definition of $Simpl(Q, A)$ without changing the necessary change of the program

$$(Simpl(Q, A) \cup \{l \leftarrow : l \in A\} \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}).$$

Thus,

$$R^c = NC(Q \cup \{l \leftarrow : l \in A\} \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}).$$

Let us show that

(4.1) $$\{l \leftarrow : l \in A\} \subseteq \{\alpha \leftarrow : \alpha \in I(I, R)\}).$$

Indeed, let $l \in A$. Then, $l \in R^c$ and $l \in I^c$. Therefore, $l \in I(I, R)$. Consequently, equation (4.1) holds. We get that

$$NC(Q \cup \{l \leftarrow : l \in A\} \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}) = NC(Q \cup \{\alpha \leftarrow : \alpha \in I(I, R)\}).$$

Hence, $R^c = NC(Q \cup \{\alpha \leftarrow : \alpha \in I(I, R)\})$. Therefore, by Theorem 2.7, R is a Q-justified revision of I. □

Theorem 4.5 *Let P be a revision program. Let I be an initial database. Let (P_k, A_k, X_k), $k \geq 0$, be the sequence of triples constructed according to the Definition 4.8. Then, for every $k \geq 0$ the following holds.*

1. *R is a P-justified revision of I if and only if R is a $(P_k \cup \{l \leftarrow : l \in A_k\})$-justified revision of I.*

2. *For every R which is a P-justified revision of I, the following holds. If $k = 2i$ then $(X_k \cup A_k) \subseteq R^c$, and if $k = 2i + 1$ then $R^c \subseteq (X_k \cup A_k \cup I^c)$.*

Proof.

By induction on k.

Basis step ($k = 0$). Both statements 1 and 2 trivially hold because $P_0 = P$, $A_0 = \emptyset$, $X_0 = \emptyset$.

Inductive step. Assume that the statement of the theorem is true for $k - 1$. We need to prove that it holds for k ($k \geq 1$).

Case 1. Let $k = 2i + 1$. By inductive hypothesis,

(1) R is a P-justified revision of I if and only if R is a $(P_{2i} \cup \{l \leftarrow : l \in A_{2i}\})$-justified revision of I, and

(2) for every R which is a P-justified revision of I, $(X_{2i} \cup A_{2i}) \subseteq R^c$.

We need to prove that

(3) R is a P-justified revision of I if and only if R is a $(P_{2i+1} \cup \{l \leftarrow : l \in A_{2i+1}\})$-justified revision of I, and

(4) for every R which is a P-justified revision of I, $R^c \subseteq (X_{2i+1} \cup A_{2i+1} \cup I^c)$.

By definition, $A'_{2i+1} = NC(P_{2i})$, $A_{2i+1} = A_{2i} \cup A'_{2i+1}$, $P_{2i+1} = Simpl(P_{2i}, A'_{2i+1})$, $X_{2i+1} = \gamma_{P_{2i+1},I}(X_{2i})$.

Proof of (3):

Consider revision program $Q = P_{2i} \cup \{l \leftarrow : l \in A_{2i}\}$. Since P_{2i} does not contain in the bodies of its rules literals from $A_{2i} \cup A^D_{2i}$, we get that $NC(Q) = NC(P_{2i} \cup \{l \leftarrow : l \in A_{2i}\}) = NC(P_{2i}) \cup A_{2i} = A'_{2i+1} \cup A_{2i} = A_{2i+1}$. Also, $P_{2i+1} = Simpl(P_{2i}, A'_{2i+1}) = Simpl(P_{2i}, A_{2i+1})$. From this and the fact that $A_{2i} \subseteq A_{2i+1}$, it follows that

$$P_{2i+1} \cup \{l \leftarrow : l \in A_{2i+1}\} =$$

$$Simpl(P_{2i}, A_{2i+1}) \cup \{l \leftarrow : l \in A_{2i+1}\} =$$

$$Simpl(P_{2i} \cup \{l \leftarrow : l \in A_{2i}\}, A_{2i+1}) \cup \{l \leftarrow : l \in A_{2i+1}\} =$$

$$Simpl(Q, NC(Q)) \cup \{l \leftarrow : l \in NC(Q)\}.$$

Therefore, by Lemma 4.4, R is a $(P_{2i} \cup \{l \leftarrow : l \in A_{2i}\})$-justified revision of I if and only if R is a $P_{2i+1} \cup \{l \leftarrow : l \in A_{2i+1}\}$-justified revision of I. Now, using the inductive assumption (1), we get statement (3).

Proof of (4):

Assume that R is a P-justified revision of I. By definition, $X_{2i+1} = \gamma_{P_{2i+1},I}(X_{2i})$. It is easy to see that

$$X_{2i+1} \cup A_{2i+1} = \gamma_{P_{2i+1},I}(X_{2i}) \cup A_{2i+1} =$$

$$\gamma_{(P_{2i+1} \cup \{l \leftarrow : l \in A_{2i+1}\}),I}(X_{2i} \cup A_{2i}).$$

Let $P' = P_{2i+1} \cup \{l \leftarrow : l \in A_{2i+1}\}$. By (3), R is a P'-justified revision of I. Since γ is anti-monotone and $(X_{2i} \cup A_{2i}) \subseteq R^c$, we get that

$$\gamma_{P',I}(R^c) \subseteq \gamma_{P',I}(X_{2i} \cup A_{2i}).$$

Hence,

$$\gamma_{P',I}(R^c) \cup I^c \subseteq \gamma_{P',I}(X_{2i} \cup A_{2i}) \cup I^c.$$

Therefore, by Lemma 4.2,

$$R^c = \gamma_{P',I}(R^c) \cup I(I,R) \subseteq \gamma_{P',I}(X_{2i} \cup A_{2i}) \cup I^c = X_{2i+1} \cup A_{2i+1} \cup I^c.$$

Case 2. $k = 2i$, $k \geq 2$. By inductive hypothesis,

(5) R is a P-justified revision of I if and only if R is a $(P_{2i-1} \cup \{l \leftarrow : l \in A_{2i-1}\})$-justified revision of I, and

(6) for every R which is a P-justified revision of I, $R^c \subseteq (X_{2i-1} \cup A_{2i-1} \cup I^c)$.

We need to prove that

(7) R is a P-justified revision of I if and only if R is a $(P_{2i} \cup \{l \leftarrow : l \in A_{2i}\})$-justified revision of I, and

(8) for every R which is a P-justified revision of I, $(X_{2i} \cup A_{2i}) \subseteq R^c$.

By definition, $A'_{2i} = \{l \in I^c : l^D \notin X_{2i-1}\ \&\ l \notin A_{2i-1}\ \&\ l^D \notin A_{2i-1}\}$, $A_{2i} = A_{2i-1} \cup A'_{2i}$, $P_{2i} = Simpl(P_{2i-1}, A'_{2i})$, $X_{2i} = \gamma_{P_{2i},I}(X_{2i-1})$.

Proof of (7):

Let $Q = P_{2i-1} \cup \{l \leftarrow : l \in A_{2i-1}\}$. Let $X = X_{2i-1} \cup A_{2i-1}$. Then, from (5) and (6) it follows that for every R which is a Q-justified revision of I, $R^c \subseteq X \cup I^c$.

Let $A = \{l \in I^c : l^D \notin X\}$. Let $Q' = Simpl(Q, A) \cup \{l \leftarrow : l \in A\}$. By Lemma 4.5, R is a Q-justified revision of I if and only if R is a Q'-justified revision of I. Using (5) we get that R is a P-justified revision of I if and only if R is a Q'-justified revision of I.

Therefore, to prove (7) we need to show that $Q' = P_{2i} \cup \{l \leftarrow : l \in A_{2i}\}$.

Indeed,

$$Q' = Simpl(Q, A) \cup \{l \leftarrow : l \in A\} =$$

$$= Simpl(P_{2i-1} \cup \{l \leftarrow : l \in A_{2i-1}\}, A) \cup \{l \leftarrow : l \in A\} =$$

$$= Simpl(P_{2i-1}, A) \cup Simpl(\{l \leftarrow : l \in A_{2i-1}\}, A) \cup \{l \leftarrow : l \in A\} =$$

$$= Simpl(P_{2i-1}, A) \cup \{l \leftarrow : l \in A_{2i-1} \setminus A\} \cup \{l \leftarrow : l \in A\} =$$

$$= Simpl(P_{2i-1}, A) \cup \{l \leftarrow : l \in A_{2i-1} \cup A\}.$$

By definition, $A = \{l \in I^c : l^D \notin X_{2i-1} \ \& \ l^D \notin A_{2i-1}\}$. Consequently, $A = A'_{2i} \cup \{l \in I^c : l^D \notin X_{2i-1} \ \& \ l^D \notin A_{2i-1} \ \& \ l \in A_{2i-1}\}$. Thus, $A_{2i-1} \cup A = A_{2i-1} \cup A'_{2i} = A_{2i}$. Hence,

$$Q' = Simpl(P_{2i-1}, A) \cup \{l \leftarrow : l \in A_{2i}\}.$$

Also, if a revision literal $l \in A_{2i-1}$, then l and l^D do not appear in the bodies of the rules from P_{2i-1}, and l does not appear in the heads of the rules from P_{2i-1}. Therefore, $Simpl(P_{2i-1}, A) = Simpl(P_{2i-1}, A \setminus A_{2i-1}) = Simpl(P_{2i-1}, A'_{2i}) = P_{2i}$. Consequently,

$$Q' = P_{2i} \cup \{l \leftarrow : l \in A_{2i}\}.$$

This finishes the proof of (7).

Proof of (8):

Assume that R is a P-justified revision of I. By definition, $X_{2i} = \gamma_{P_{2i},I}(X_{2i-1})$. It is easy to see that

$$X_{2i} \cup A_{2i} = \gamma_{P_{2i},I}(X_{2i-1}) \cup A_{2i} = \gamma_{(P_{2i} \cup \{l \leftarrow : l \in A_{2i}\}),I}(X_{2i-1} \cup A_{2i-1}).$$

Let $P' = P_{2i} \cup \{l \leftarrow : l \in A_{2i}\}$. By (7), R is a P'-justified revision of I. Since γ is anti-monotone and $R^c \subseteq (X_{2i-1} \cup A_{2i-1} \cup I^c)$, we get that

$$\gamma_{P',I}(R^c) \supseteq \gamma_{P',I}(X_{2i-1} \cup A_{2i-1} \cup I^c).$$

From the definition of γ it follows that

$$\gamma_{P',I}(X_{2i-1} \cup A_{2i-1} \cup I^c) = \gamma_{P',I}(X_{2i-1} \cup A_{2i-1}).$$

Hence,

$$\gamma_{P',I}(R^c) \supseteq \gamma_{P',I}(X_{2i-1} \cup A_{2i-1}) = X_{2i} \cup A_{2i}.$$

By Lemma 4.2, $R^c = \gamma_{P',I}(R^c) \cup I(I, R)$. Therefore,

$$R^c \supseteq \gamma_{P',I}(R^c) \supseteq X_{2i} \cup A_{2i}.$$

That is, (8) holds. □

Corollary 4.2 *Let R be a P-justified revision of I. Then, for every $k \geq 0$, $R \models A_k$.*

Proof.
By Theorem 4.5, R is a $(P_k \cup \{l \leftarrow : l \in A_k\})$-justified revision of I. Hence, $R \models A_k$. □

Corollary 4.3 *If in the sequence of triples (P_k, A_k, X_k) there exists k such that A_k is incoherent, then there are no P-justified revisions of I.*

Proof.
Follows from Corollary 4.2. □

Lemma 4.6 *In the Definition 4.8, $X_{2i} \subseteq A'_{2i+1}$ for $i \geq 0$.*

Proof.
If $i = 0$, then $X_0 = \emptyset$. Thus, $X_0 \subseteq A'_1$.

Let $i \geq 1$. Then, by definition, $A'_{2i+1} = NC(P_{2i})$, and $X_{2i} = \gamma_{P_{2i},I}(X_{2i-1})$. By definition of operator γ, $X_{2i} = NC((P_{2i}, I)^{X_{2i-1}})$. By definition of the reduct, $(P_{2i}, I)^{X_{2i-1}}$ is obtained from P_{2i} by

1. removing every rule $r \in P_{2i}$ such that $body(r) \cap \{l^D : l \in X_{2i-1} \setminus I^c\} \neq \emptyset$,

2. deleting from the body of each remaining rule any revision literal that is in I^c.

Therefore, for every rule $r \in (P_{2i}, I)^{X_{2i-1}}$, there is a rule $r' \in P_{2i}$, such that $head(r) = head(r')$ and $body(r) \subseteq body(r')$. We will prove that in fact, $body(r) = body(r')$.

Assume the contrary. Assume that $l \in body(r')$ and $l \notin body(r)$. Then, by the second step of the definition of the reduct, $l \in I^c$. Hence, $l^D \notin I^c$.

Let us show that $l^D \in X_{2i-1}$. Assume the contrary, that $l^D \notin X_{2i-1}$. Because of the properties of $Simpl$ transformation, if $l \in A_{2i-1}$ or $l^D \in A_{2i-1}$, then l can not be in the body of a rule in P_{2i}. However, $l \in body(r')$, where $r' \in P_{2i}$. Therefore, $l \notin A_{2i-1}$ and $l^D \notin A_{2i-1}$. Then, by construction of triples (P_k, A_k, X_k), $l \in \{l \in I^c : l^D \notin X_{2i-1} \& l \notin A_{2i-1} \& l^D \notin A_{2i-1}\}$. That is, $l \in A'_{2i}$. Hence, by properties of $Simpl$ transformation, l can not be in the body of a rule from P_{2i}. This contradicts the fact that $l \in body(r')$ and $r' \in P_{2i}$. Consequently, $l^D \in X_{2i-1}$.

We have: $l^D \notin I^c$ and $l^D \in X_{2i-1}$. Thus, $l^D \in X_{2i-1} \setminus I^c$. At the same time, $l \in body(r')$. According to step 1 in the definition of the reduct $(P_{2i}, I)^{X_{2i-1}}$, rule r' will be removed. This contradicts the fact that $r' \in P_{2i}$. Therefore, for every rule $r \in (P_{2i}, I)^{X_{2i-1}}$, there is a rule $r' \in P_{2i}$, such that $head(r) = head(r')$ and $body(r) = body(r')$. Hence, $NC((P_{2i}, I)^{X_{2i-1}}) \subseteq NC(P_{2i})$. In other words, $X_{2i} \subseteq A'_{2i+1}$. □

Corollary 4.4 *In Definition 4.8, for every $i \geq 0$, $X_{2i} \cup A_{2i} \subseteq X_{2i+2} \cup A_{2i+2}$.*

Proof.
By Definition 4.8, $A_{2i} \subseteq A_{2i+2}$. By Lemma 4.6, $X_{2i} \subseteq A_{2i+1} \subseteq A_{2i+2}$. Hence, $X_{2i} \cup A_{2i} \subseteq X_{2i+2} \cup A_{2i+2}$. □

4.2.2 Comparison with definitions induced by embeddings

The following is an example when the new definition of well-founded semantics gives a better result than WFS^{Sh}.

Example 4.6 *Let $I = \emptyset$. Consider revision program from Example 4.3*

$$\begin{aligned} P: \quad \mathbf{in}(a) &\leftarrow \mathbf{out}(b) \\ \mathbf{in}(b) &\leftarrow \mathbf{out}(a) \\ \mathbf{out}(b) &\leftarrow \end{aligned}$$

We have the following computation:

$$A_0 = \emptyset, P_0 = P, X_0 = \emptyset \mapsto$$

$$\mapsto A_1 = \left\{ \begin{array}{c} \mathbf{out}(b), \\ \mathbf{in}(a) \end{array} \right\}, P_1 = \emptyset, X_1 = \emptyset.$$

The WFS^{Sh} for P was found in Example 4.3. It is less informative than the new definition of WFS.

The following is an example when the new definition of well-founded semantics gives better result than WFS^{PT}.

Example 4.7 *Let $I = \emptyset$. Consider revision program from Example 4.4*

$$\begin{array}{rrcl} P: & \mathbf{in}(a) & \leftarrow & \mathbf{out}(b) \\ & \mathbf{in}(b) & \leftarrow & \mathbf{out}(a) \\ & \mathbf{in}(c) & \leftarrow & \mathbf{out}(d) \\ & \mathbf{in}(d) & \leftarrow & \mathbf{out}(c), \mathbf{out}(f) \\ & \mathbf{out}(f) & \leftarrow & \mathbf{in}(a), \mathbf{in}(b) \\ & \mathbf{in}(f) & \leftarrow & \\ & \mathbf{in}(a) & \leftarrow & \mathbf{in}(c), \mathbf{in}(f) \end{array}$$

We have the following computation:

$$A_0 = \emptyset, P_0 = P, X_0 = \emptyset \mapsto$$

$$\mapsto A_1 = \left\{ \mathbf{in}(f) \right\}, P_1 = \left\{ \begin{array}{rcl} \mathbf{in}(a) & \leftarrow & \mathbf{out}(b) \\ \mathbf{in}(b) & \leftarrow & \mathbf{out}(a) \\ \mathbf{in}(c) & \leftarrow & \mathbf{out}(d) \\ \mathbf{out}(f) & \leftarrow & \mathbf{in}(a), \mathbf{in}(b) \\ \mathbf{in}(a) & \leftarrow & \mathbf{in}(c) \end{array} \right\}, X_1 = \left\{ \begin{array}{c} \mathbf{in}(a), \\ \mathbf{in}(b), \\ \mathbf{in}(c), \\ \mathbf{out}(f) \end{array} \right\} \mapsto$$

$$\mapsto A_2 = \left\{ \begin{array}{l} \mathbf{in}(f), \\ \mathbf{out}(d) \end{array} \right\}, P_2 = \left\{ \begin{array}{lll} \mathbf{in}(a) & \leftarrow & \mathbf{out}(b) \\ \mathbf{in}(b) & \leftarrow & \mathbf{out}(a) \\ \mathbf{in}(c) & \leftarrow & \\ \mathbf{out}(f) & \leftarrow & \mathbf{in}(a), \mathbf{in}(b) \\ \mathbf{in}(a) & \leftarrow & \mathbf{in}(c) \end{array} \right\}, X_2 = \left\{ \begin{array}{l} \mathbf{in}(c), \\ \mathbf{in}(a) \end{array} \right\} \mapsto$$

$$\mapsto A_3 = \left\{ \begin{array}{l} \mathbf{in}(f), \\ \mathbf{out}(d), \\ \mathbf{in}(c), \\ \mathbf{in}(a) \end{array} \right\}, P_3 = \left\{ \begin{array}{lll} \mathbf{out}(f) & \leftarrow & \mathbf{in}(b) \end{array} \right\}, X_3 = \emptyset \mapsto$$

$$\mapsto A_4 = \left\{ \begin{array}{l} \mathbf{in}(f), \\ \mathbf{out}(d), \\ \mathbf{in}(c), \\ \mathbf{in}(a), \\ \mathbf{out}(b) \end{array} \right\}, P_4 = \emptyset, X_4 = \emptyset.$$

The WFS^{PT} for P was found in Example 4.4. It is less informative than the new definition of WFS.

Chapter 5

Extensions of Revision Programming

The connection between revision programming and logic programming described in Section 3.2.3 naturally led to extensions of revision programming formalism which correspond to existing extensions of logic programming (general disjunctive logic programs of [16] and nested logic programs of [15]). In Section 5.1 we show that revision programs are equivalent to a special class of general logic programs, and introduce disjunctive revision programs. Results of Section 5.1 were published in [18]. In Section 5.2 we present nested revision programs.

5.1 Disjunctive revision programs.

In this section we first describe a formalism of general disjunctive logic programs ([16]). Next, we show that revision programs are equivalent to general disjunctive logic programs which have only one atom in the heads of their rules. Finally, we introduce disjunctive revision programs.

5.1.1 General disjunctive logic programs.

Lifschitz and Woo [16] introduced a formalism called general logic programming (see also [14] and [28]). General logic programming deals with clauses whose heads are 'disjunctions' of atoms (we will restrict our attention 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:

$$A_1 | \ldots | A_k | not\ A_{k+1} | \ldots | not\ A_l \ \leftarrow \ A_{l+1}, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n, \tag{5.1}$$

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 (5.1), 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. This coincides with the classical notion of satisfiability.

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

(5.2) $$A_1 | \ldots | A_k \leftarrow A_{k+1}, \ldots, A_m$$

Given a program P consisting of clauses of type (5.2), 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.

5.1.2 Answer sets for general programs and justified revisions

We will now show that revision programming is closely related to 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 *unitary*. A *unitary general logic program* consists of unitary rules.

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:

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

Notice that this is just the converse to translation in Section 2.4.3.

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 $dj(P) = \{dj(r)\colon r \in P\}$.

This mapping is a 1-1 correspondence between 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 2.14.

Theorem 5.1 *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 $dj(P)$.*

Proof.

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

Observe that R is P-justified revision of $\emptyset$ if and only if R is P'-justified revision of $\emptyset$. Indeed, inertia $I(\emptyset, R) = \{\mathbf{out}(a) : a \notin R\}$. From Theorem 2.7 we have that R is P-justified revision of $\emptyset$ if and only if

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

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

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

Therefore, using Theorem 2.7 again, we get that R is P-justified revision of $\emptyset$ if and only if R is P'-justified revision of $\emptyset$.

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

Let $P'_{\emptyset,R} = P'' \cup P'''$, where P'' consists of all in-rules of P', $P''' = P' \setminus P''$ consists of all out-rules of P'. 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 $a \in R$. By definition, $dj(P)^R$ is obtained from $dj(P')$ by replacing each disjunctive rule by $HPos \leftarrow BPos$. Therefore,

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

After these observations we are ready to prove the statement of the theorem.

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

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

It might appear that the scope of Theorem 5.1 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 5.1 *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. Recall that 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 5.2 *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'.*

Proof.
By the shifting theorem it is enough to prove the statement for the case $I = \emptyset$. Since $I = \emptyset$, P' consists of all out-rules of P and P'' consists of all in-rules of P.
$(\Rightarrow)$ If R is a P-justified revision of $\emptyset$, then R is a model of P. Hence, it is a model of $P' \subseteq P$. Theorem 2.7 implies that

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

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

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

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

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

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

If we remove from the program some rules with premises that 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, $head(r) \in \{\textbf{out}(a) : a \in R\}$. Assume that premises of r

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

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

From Theorem 2.7 we conclude that R is a P''-justified revision of $\emptyset$.

($\Leftarrow$) Assume R is a P''-justified revision of $\emptyset$, and R is a model of P'. Theorem 2.7 implies that

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

Let $M = \{\mathbf{in}(a) : a \in R\} \cup \{\mathbf{out}(a) : a \notin R\}$. By the definition of necessary change, M is the least model of

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

Thus, 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

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

Hence,

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

By Theorem 2.7, R 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 eliminate some of the computed revisions.

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 [14]). Theorem 5.2 has the following corollary.

Corollary 5.2 *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 5.1 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 act as constraints.

Corollary 5.3 *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 $dj(T_I(P) \setminus P')$ that satisfies P', where P' consists of all constraints in $T_I(P)$.*

5.1.3 Disjunctive revision programs.

The results of Section 5.1.2 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 5.1.2 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:

$$\alpha_1 | \ldots | \alpha_m \leftarrow \alpha_{m+1}, \ldots, \alpha_n \ , \tag{5.3}$$

where α_i, $1 \leq i \leq n$, are *revision 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 revision literals under a disjunctive rule. A set L of literals is *closed* under a rule (5.3) 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 revision literals that is closed under P. Notice that in the context of disjunctive programs the necessary change may be not 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).

Definition 5.1 *The* reduct *of a disjunctive revision program P with respect to I and R, 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.* $\triangle$

We are now ready to define the notion of 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 5.1 Consider Example 2.2, where revision program is represented as disjunctive revision program P:

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

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

The following three theorems show that the semantics for disjunctive revision programs described here satisfies the three criteria described at the beginning of this subsection.

Theorem 5.3 *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.*

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 **in**(a) and **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 5.4 (Shifting theorem for disjunctive revision programming) *Let I_1 and I_2 be databases, and P be a disjunctive revision program. Let $W = I_1 \div I_2$. Then R_1 is a P-justified revision of I_1 if and only if $T_W(R_1)$ is $T_W(P)$-justified revision of I_2.*

Proof.
Similarly to the case of ordinary revision programs, in computing justified revisions for disjunctive revision programs we are dealing with revision 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 **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 5.5 *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 $dj(P)$.*

Proof.

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

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

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

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

Step 4 removes from the heads of the rules all literals that contradict R. After step 2 all remaining rules r satisfy the condition that $dj(r)$ has $HNeg \subseteq R$. Hence, step 4 corresponds to eliminating all $not(HNeg)$ parts and all other literals that contradict R from the heads of the remaining rules in the disjunctive logic program.

Therefore, $dj(P)^R$, when compared to $dj(P^{I,R})$, may only have some extra rules, the bodies of which are not satisfied by R, or some extra literals in the heads, which are not satisfied by 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. Disjunctive revision programming is an essential extension of unitary revision programming. The existence problem for P-justified revisions is NP-complete (Theorem 2.13). Eiter and Gottlob showed that the existence problem for answer sets for general logic programs is Σ_2^P-complete ([7]). Using the results of Eiter and Gottlob [7] and our correspondence between disjunctive revision programs and general logic programs we obtain the following result.

Theorem 5.6 *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.

5.2 Nested revision programs.

In this section we describe nested expressions in logic programs from [15]. Then, we introduce nested expressions in revision programs.

5.2.1 Nested expressions in logic programs

Lifschitz, Tang and Turner ([15]) extended the answer set semantics to a class of logic programs with nested expressions permitted in the bodies and heads of rules. The following are basic definitions from that paper.

Literals are atoms and classical negations of atoms.

Elementary formulas are literals and the symbols $\bot$ ("false") and $\top$ ("true"). Formulas are built from elementary formulas using the unary connective not and the binary connectives , (conjunction) and ; (disjunction). A $rule$ is an expression of the form

$$F \leftarrow G$$

where F and G are formulas, called the $head$ and the $body$ of the rule.

A $program$ is a set of rules.

The formulas, rules and programs that do not contain the negation as failure operator not are called $basic$.

A set X of literals $satisfies$ a basic formula F (symbolically, $X \models F$) when:

- for elementary F, $X \models F$ if $F \in X$, or $F = \top$, or X contains a complementary pair.
- $X \models (F, G)$ if $X \models F$ and $X \models G$.
- $X \models (F; G)$ if $X \models F$ or $X \models G$.

Let Π be a basic program. A set X of literals is $closed$ under Π if, for every rule $F \leftarrow G$ in Π, $X \models F$ whenever $X \models G$. X is an $answer\ set$ for Π if X is a minimal set of literals that is both logically closed and closed under Π.

The $reduct$ of a formula, rule or program relative to a set X of literals is defined recursively, as follows:

- for elementary F, $F^X = F$.
- $(F, G)^X = (F^X, G^X)$.
- $(F; G)^X = (F^X; G^X)$.
- $(not\ F)^X = \begin{cases} \bot, & \text{if } X \models F^X \\ \top, & \text{otherwise} \end{cases}$
- $(F \leftarrow G)^X = F^X \leftarrow G^X$.
- $\Pi^X = \{(F \leftarrow G)^X : F \leftarrow G \in \Pi\}$.

A set X of literals is an $answer\ set$ for Π if it is an answer set for the reduct Π^X.

5.2.2 Nested expressions in revision programs

We can extend justified revision semantics to a class of revision programs with nested expressions permitted in the rules. These expressions are formed from literals using the binary operators , and ; and unary operators in, out that can be nested arbitrarily.

Elementary formulas are literals and the symbols $\bot$ ("false") and $\top$ ("true"). Recall that a 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.

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 $basic$.

A set X of literals $satisfies$ a basic formula F (denoted $X \models F$) if

- $F = \top$
- F is literal and $F \in X$
- $F = (A, B)$ for some formulas A and B, $X \models A$ and $X \models B$
- $F = (A; B)$ for some formulas A and B, $X \models A$ or $X \models B$

Definition 5.2 *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 $X \models B$.* △

Definition 5.3 *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 literals that is closed under P.* △

Definition 5.4 *The $reduct$ of a formula, rule, and program relative to a set X of atoms (denoted F^X) is defined recursively, as follows:*

- *for elementary F, $F^X = F$.*
- $(F, G)^X = (F^X, G^X)$.
- $(F; G)^X = (F^X; G^X)$.
- $(out\ F)^X = \begin{cases} \bot, & \text{if } X^c \models F^X \\ \top, & \text{otherwise} \end{cases}$
- $(in\ F)^X = \begin{cases} \bot, & \text{if } X^c \not\models F^X \\ \top, & \text{otherwise} \end{cases}$
- $(F \leftarrow G)^X = F^X \leftarrow G^X$
- $P^X = \{(F \leftarrow G)^X : F \leftarrow G \in P\}$ △

Definition 5.5 *Let I and R be two databases. Let Y be a formula, rule or program. A $reduct$ of Y with respect to I, R (denoted $Y^{(I,R)}$) is obtained from Y^R by replacing each occurrence of literals from $I(I, R)$ by $\top$, and replacing each occurrence of literals from $I(I, R)^D$ by $\bot$.* △

Definition 5.6 *Let P be a nested revision program. A database R is a $P\text{-}justified\ revision$ of a database I if for some necessary change L of $P^{(I,R)}$, L is coherent and $R = I \oplus L$.* △

Theorem 5.7 *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.*

Proof.

Let us compare reducts $P^{I,R}$ and $P^{(I,R)}$.

Since P is a basic revision program, $P^{(I,R)}$ is obtained from P by replacing each occurence of literals from $I(I,R)$ by $\top$, and replacing each occurence of literals from $I(I,R)^D$ by $\bot$.

During Step 2 of constructing reduct $P^{I,R}$, all literals from inertia $I(I,R)$ are eliminated from the bodies of the rules. From the point of view of satisfiability it has the same effect as replacing them with $\top$. That is, if B is a body of a rule (B is a conjunction of literals), B' is obtained from B by eliminating all literals from inertia, B'' is obtained from B by replacing all literals from inertia by $\top$, and X is a set of literals, then $B' \subseteq X$ if and only if $X \models B''$.

In Step 3 of constructing reduct $P^{I,R}$ all rules with literals from $I(I,R)$ in the heads are eliminated. That means that rules with literals from inertia in the heads are not used in computing the necessary change. Replacing literals from $I(I,R)$ in the heads of the rules by $\top$ has the same effect. Indeed, if a rule r has $\top$ in the head (recall that head of r is a disjunction of literals), then any set of literals is closed under r.

In Step 1 of constructing reduct $P^{I,R}$ all rules whose body is not satisfied by R are eliminated. This has no effect on computing necessary changes. Indeed, necessary changes will not satisfy the bodies of eliminated rules, therefore they will be closed under these rules.

In step 4 of constructing reduct $P^{I,R}$ all literals that contradict R are removed from the heads. If R is P-justified revision of I, then by definition, $R = I \oplus L$, where L is some necessary change. Hence, literals that contradict R are not in L. If we add or remove literals that are not in L from the heads of disjunctive rules, L will remain the necesary change of a modified in such a way program.

Replacing each occurence of literals from $I(I,R)^D$ by $\bot$ also has no effect on computing necessary changes. Indeed, literals from $I(I,R)^D$ contradict R. Thus, they will not be in necessary change, and can be replaced by $\bot$.

Therefore, the statement of the theorem holds. □

Theorem 5.8 (Shifting theorem for nested revision programming) *Let I_1 and I_2 be databases. Let P be a nested revision program. Then, R_1 is P-justified revision of I_1 if and only if $T_{I_1 \div I_2}(R_1)$*

is a $T_{I_1 \div I_2}(P)$-justified revision of I_2.

Proof.

Similar to the proof of the Shifting Theorem in ordinary revision program case (Theorem 3.4). □

Given a nested revision program P we define a nested logic program $nlp(P)$ 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 conective in by $not\ not$.

Theorem 5.9 *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 $nlp(P)$.*

Proof.

The proof is based on the following lemma.

Lemma 5.1 *Let M be a coherent set of literals, $N \subseteq M$. Let F be a formula (in revision programming sence). Then $N \models F^{(\emptyset, M^+)}$ if and only if $N^+ \models nlp(F)^{M^+}$.*

Proof by induction on the depth of the formula F. □

Proof of the theorem:

($\Rightarrow$) Assume that R is an answer set for $nlp(P)$. It means that R is a minimal set of literals closed under $nlp(P)^R$. By the lemma, $\{in(a) : a \in R\}$ is closed under $P^{(\emptyset, M)}$. To show that R is a P-justified revision of $\emptyset$ we need to show that $\{in(a) : a \in R\}$ is minimal set that is closed under $P^{(\emptyset, M)}$.

Assume the contrary. Let $L \subset \{in(a) : a \in R\}$ be closed under $P^{(\emptyset, M)}$. Then $L^+ \subset R$. By Lemma 5.1, L^+ is closed under $nlp(P)^R$. Contradiction.

($\Leftarrow$) Similarly to the previous case. □

The following example illustrates a possible use of unary connective in in nested revision programs.

Example 5.2 Consider nested revision programs $P = \{in(q) \leftarrow\}$ and $P' = \{in(in(q)) \leftarrow\}$. It is easy to see that every database has a P-justified revision. However, only databases which contain q have a P'-justified revision.

Chapter 6

Annotated Revision programs

This chapter studies annotated revision programs. The results from this chapter were published in [17], [19].

6.1 Introducing annotations.

We will start with examples that illustrate the main notions and a possible use of annotated revision programming. Formal definitions will follow.

Example 6.1 A group of experts is about to discuss a certain proposal and then vote whether to accept or reject it. Each person has an opinion on the proposal that may be changed during the discussion as follows:

- any person can convince an optimist to vote for the proposal,

- any person can convince a pessimist to vote against the proposal.

The group consists of two optimists (Ann and Bob) and one pessimist (Pete). We want to be able to answer the following question: given everybody's opinion on the subject before the discussion, what are the possible outcomes of the vote?

Assume that before the vote Pete is for the proposal, Bob is against, and Ann is indifferent (has no arguments for and no arguments against the proposal). This situation can be described by assigning to atom "$accept$" the annotation $\langle\{Pete\}, \{Bob\}\rangle$, where the first element of the pair is the set of experts who have arguments for the acceptance of the proposal and the second element is the set of experts who have arguments against the proposal. In the formalism of annotated revision programs, as proposed by Fitting in [9], this initial situation is described by a *function* that assigns to each atom in the language (in this example there is only one atom) its annotation. In our example, this function is given by: $B_I(accept) = \langle\{Pete\}, \{Bob\}\rangle$. (Let us mention here that in general, the sets of experts in an annotation need not be disjoint. An expert may have arguments for and against the proposal at the same time. In such a case the expert is contradictory.)

The ways in which opinions may change are described by the following annotated revision rules:

$$
\begin{aligned}
(\mathbf{in}(accept){:}\{Ann\}) &\leftarrow (\mathbf{in}(accept){:}\{Bob\})\\
(\mathbf{in}(accept){:}\{Ann\}) &\leftarrow (\mathbf{in}(accept){:}\{Pete\})\\
(\mathbf{in}(accept){:}\{Bob\}) &\leftarrow (\mathbf{in}(accept){:}\{Ann\})\\
(\mathbf{in}(accept){:}\{Bob\}) &\leftarrow (\mathbf{in}(accept){:}\{Pete\})\\
(\mathbf{out}(accept){:}\{Pete\}) &\leftarrow (\mathbf{out}(accept){:}\{Ann\})\\
(\mathbf{out}(accept){:}\{Pete\}) &\leftarrow (\mathbf{out}(accept){:}\{Bob\})
\end{aligned}
$$

The first rule means that if Bob accepts the proposal, then Ann should accept the proposal, too, since she will be convinced by Bob. Similarly, the second rule means that if Pete has arguments for the proposal, then he will be able to convince Ann. These two rules describe Ann being an optimist. The remaining rules follow as Bob is an optimist and Pete is a pessimist.

Possible outcomes of the vote are given by justified revisions. In this particular case there are two justified revisions of the initial database B_I. They are $B_R(accept) = \langle\{Ann, Bob, Pete\}, \{\}\rangle$ and $B'_R(accept) = \langle\{\}, \{Bob, Pete\}\rangle$. The first one corresponds to the case when the proposal is accepted (Ann, Bob and Pete all voted for). This outcome happens if Pete convinces Bob and Ann to vote for. The second revision corresponds to the case when Bob and Pete voted against the proposal (Ann remained indifferent and did not vote). This outcome happens if Bob convinces Pete to change his opinion. □

Remark 6.1 *It is possible to rewrite annotated revision rules from Example 6.1 as ordinary revision rules (without annotations) if we use atoms "$accept\_Ann$", "$accept\_Bob$", and "$accept\_Pete$". However, ordinary revision programs do not deal with inconsistent or not completely defined databases. In particular, we will not be able to express the fact that initially Ann has no arguments for and no arguments against the proposal in Example 6.1.*

In the next example annotations are numbers from 0 to 1 representing different degrees of a particular quality.

Example 6.2 Assume that there are two sources of light: a and b. Each of them may be either On or Off. They are used to transmit two signals. The first signal is a combination of a being On and b being Off. The second signal is a combination of a being Off and b being On.

The sources a and b are located far from an observer. Such factors as light pollution and dust may affect the perception of signals. Therefore, the observed brightness of a light source differs from its actual brightness. Assume that brightness is measured on a scale from 0 (complete darkness) to 1 (maximal brightness). The actual brightness of a light source may be either 0 (when it is Off), or 1 (when it is On).

Initial database B_I represents observed brightness of sources. For example, if observed brightness of source a is α ($0 \leq \alpha \leq 1$), then $B_I(a) = \langle \alpha, 1-\alpha \rangle$. We may think of the first and the second elements in the pair $\langle \alpha, 1-\alpha \rangle$ as degrees of brightness and darkness of the source respectively. The task is to infer actual brightness from observed brightness. Thus, revision of the initial database should represent actual brightness of sources.

Suppose we know that dust in the air can not reduce brightness by more than 0.2. Then, we can safely presume that a light source is On if its observed brightness is 0.8 or more. Assume also that light pollution can not contribute more than 0.4. That is, if observed darkness of a source is at least 0.6, the light must be Off. This information together with the fact that only two signals are possible, may be represented by the following annotated revision program P:

$$
\begin{aligned}
(\mathbf{in}(a){:}1) &\leftarrow (\mathbf{in}(a){:}0.8), (\mathbf{out}(b){:}0.6) \\
(\mathbf{out}(b){:}1) &\leftarrow (\mathbf{in}(a){:}0.8), (\mathbf{out}(b){:}0.6) \\
(\mathbf{in}(b){:}1) &\leftarrow (\mathbf{in}(b){:}0.8), (\mathbf{out}(a){:}0.6) \\
(\mathbf{out}(a){:}1) &\leftarrow (\mathbf{in}(b){:}0.8), (\mathbf{out}(a){:}0.6)
\end{aligned}
$$

The first two rules state that if the brightness of a is at least 0.8 and darkness of b is at least 0.6, then brightness of a is 1 (the first rule) and darkness of b is 1 (the second rule). This corresponds to the case when the first signal is transmitted. Similarly, the last two rules describe the case when the second signal is transmitted.

Let observed brightness of a and b be 0.3 and 0.9 respectively. That is, $B_I(a) = \langle 0.3, 0.7 \rangle$ and $B_I(b) = \langle 0.9, 0.1 \rangle$. Then, P-justified revision of B_I is the actual brightness. It is $B_R(a) = \langle 0, 1 \rangle$, and $B_R(b) = \langle 1, 0 \rangle$. □

Now let us move on to formal definitions. Let U be a fixed *universe* whose elements are referred to as *atoms*. In Example 6.1 $U = \{accept\}$. In Example 6.2 $U = \{a, b\}$. Expressions of the form $\mathbf{in}(a)$ and $\mathbf{out}(a)$, where $a \in U$, are called *revision atoms*. Revision atoms are assigned annotations. These annotations are members of a *complete infinitely distributive lattice with the de Morgan complement* (an order reversing involution). We denote this lattice by $\mathcal{T}$. The partial ordering on $\mathcal{T}$ is denoted by $\leq$ and the corresponding meet and join operations by $\wedge$ and $\vee$, respectively. The de Morgan complement of $a \in \mathcal{T}$ is denoted by $\overline{a}$. Let us recall that it satisfies the following two laws (the de Morgan laws):

$$\overline{a \vee b} = \overline{a} \wedge \overline{b}, \qquad \overline{a \wedge b} = \overline{a} \vee \overline{b}.$$

In Example 6.1, $\mathcal{T}$ is the set of subsets of the set $\{Ann, Bob, Pete\}$, with $\subseteq$ as the ordering relation, and the set-theoretic complement as the de Morgan complement. In Example 6.2, $\mathcal{T} = [0, 1]$ with the usual ordering; the de Morgan complement of α is $1 - \alpha$.

An *annotated revision atom* is an expression of the form $(\mathbf{in}(a){:}\alpha)$ or $(\mathbf{out}(a){:}\alpha)$, where $a \in U$ and $\alpha \in \mathcal{T}$. An *annotated revision rule* is an expression of the form

$$p \leftarrow q_1, \ldots, q_n,$$

where p, $q_1, \ldots, q_n$ are annotated revision atoms. An *annotated revision program* is a set of annotated revision rules.

A *$\mathcal{T}$-valuation* is a mapping from the set of revision atoms to $\mathcal{T}$. A $\mathcal{T}$-valuation v describes our information about the membership of the elements from U in some (possibly unknown) set $B \subseteq U$. For instance, $v(\mathbf{in}(a)) = \alpha$ can be interpreted as saying that $a \in B$ with certainty α. A $\mathcal{T}$-valuation v *satisfies* an annotated revision atom $(\mathbf{in}(a){:}\alpha)$ if $v(\mathbf{in}(a)) \geq \alpha$. Similarly, v *satisfies* $(\mathbf{out}(a){:}\alpha)$ if $v(\mathbf{out}(a)) \geq \alpha$. The $\mathcal{T}$-valuation v *satisfies* a list or a set of annotated revision atoms if it satisfies each member of the list or the set. A $\mathcal{T}$-valuation *satisfies* an annotated revision rule if it satisfies the head of the rule whenever it satisfies the body of the rule. Finally, a $\mathcal{T}$-valuation *satisfies* an annotated revision program (is a *model* of the program) if it satisfies all rules in the program.

Given an annotated revision program P we can assign to it an operator on the set of all $\mathcal{T}$-valuations. Let $t_P(v)$ be the set of the heads of all rules in P whose bodies are satisfied by a

$\mathcal{T}$-valuation v. We define an operator T_P as follows:

$$T_P(v)(l) = \bigvee\{\alpha | (l{:}\alpha) \in t_P(v)\}$$

Here $\bigvee X$ is the join of the subset X of the lattice (note that $\bot$ is the join of an empty set of lattice elements). The operator T_P is a counterpart of the well-known van Emden-Kowalski operator from logic programming.

It is clear that under $\mathcal{T}$-valuations, the information about an element $a \in U$ is given by a pair of elements from $\mathcal{T}$ that are assigned to revision atoms **in**(a) and **out**(a). Thus, we will also consider an algebraic structure $\mathcal{T}^2$ with the domain $\mathcal{T} \times \mathcal{T}$ and with an ordering $\leq_k$ defined by:

$$\langle \alpha_1, \beta_1 \rangle \leq_k \langle \alpha_2, \beta_2 \rangle \quad \text{if} \quad \alpha_1 \leq \alpha_2 \text{ and } \beta_1 \leq \beta_2.$$

If a pair $\langle \alpha_1, \beta_1 \rangle$ is viewed as a measure of our information about membership of a in some unknown set B then $\alpha_1 \leq \alpha_2$ and $\beta_1 \leq \beta_2$ imply that the pair $\langle \alpha_2, \beta_2 \rangle$ represents higher degree of knowledge about a. Thus, the ordering $\leq_k$ is often referred to as the *knowledge* or *information* ordering. Since the lattice $\mathcal{T}$ is complete and distributive, $\mathcal{T}^2$ is a complete distributive lattice with respect to the ordering $\leq_k$.

Remark 6.2 *There is another ordering that can be associated with $\mathcal{T}^2$. We can define $\langle \alpha_1, \beta_1 \rangle \leq_t \langle \alpha_2, \beta_2 \rangle$ if $\alpha_1 \leq \alpha_2$ and $\beta_1 \geq \beta_2$. This ordering is often called the* truth ordering. *Since $\mathcal{T}$ is a complete distributive lattice, $\mathcal{T}^2$ with both orderings $\leq_k$ and $\leq_t$ forms a complete distributive* bilattice *(see [11, 8] for a definition). However we will not use the ordering $\leq_t$ nor the fact that $\mathcal{T}^2$ is a bilattice.*

The operations of meet, join, top, and bottom under $\leq_k$ are denoted $\otimes$, $\oplus$, $\top$, and $\bot$, respectively. In addition, we make use of the *conflation* operation. Conflation is defined as $-\langle \alpha, \beta \rangle = \langle \overline{\beta}, \overline{\alpha} \rangle$. An element $A \in \mathcal{T}^2$ is *consistent* if $A \leq_k -A$. In other words, an element $\langle \alpha, \beta \rangle \in \mathcal{T}^2$ is consistent if α is smaller than or equal to the complement of β (the evidence "for" is less than or equal than the complement of the evidence "against") and β is smaller than or equal to the complement of α (the evidence "against" is less than or equal than the complement of the evidence "for").

The conflation operation satisfies the de Morgan laws:

$$-(\langle \alpha, \beta \rangle \oplus \langle \gamma, \delta \rangle) = -\langle \alpha, \beta \rangle \otimes -\langle \gamma, \delta \rangle,$$

$$-(\langle \alpha, \beta \rangle \otimes \langle \gamma, \delta \rangle) = -\langle \alpha, \beta \rangle \oplus -\langle \gamma, \delta \rangle,$$

where $\alpha, \beta, \gamma, \delta \in \mathcal{T}$.

A $\mathcal{T}^2$-valuation is a mapping from *atoms* of the universe to elements of $\mathcal{T}^2$. If $B(a) = \langle \alpha, \beta \rangle$ under some $\mathcal{T}^2$-valuation B, we say that under B the atom a is in a set with certainty α and it is not in the set with certainty β. We say that a $\mathcal{T}^2$-valuation is *consistent* if it assigns a consistent element of $\mathcal{T}^2$ to every atom in U.

We will use $\mathcal{T}^2$-valuations to represent current information about sets (databases) as well as the change that needs to be enforced. Let B be a $\mathcal{T}^2$-valuation representing our knowledge about a certain set and let C be a $\mathcal{T}^2$-valuation representing change that needs to be applied to B. We define the revision of B by C, say B', by

$$B' = (B \otimes -C) \oplus C.$$

The intuition is as follows. After the revision, the new valuation must contain at least as much knowledge about atoms being in and out as C. On the other hand, this amount of knowledge must not exceed implicit bounds present in C and expressed by $-C$, unless C directly implies so. In other words, if $C(a) = \langle \alpha, \beta \rangle$, then evidence for $\mathbf{in}(a)$ must not exceed $\bar{\beta}$ unless $\alpha \geq \bar{\beta}$, and the evidence for $\mathbf{out}(a)$ must not exceed $\bar{\alpha}$ unless $\beta \geq \bar{\alpha}$. Since we prefer explicit evidence of C to implicit evidence expressed by $-C$, we perform the change by first using $-C$ and then applying C. However, let us note here that the order matters only if C is inconsistent; if C is consistent, $(B \otimes -C) \oplus C = (B \oplus C) \otimes -C$. This specification of how the change modeled by a $\mathcal{T}^2$-valuation is enforced plays a key role in the definition of justified revisions in Section 6.3.1.

Example 6.3 (continuation of Example 6.1) In Example 6.1, B_I has two revisions. The first one, B_R, is the revision of B_I by C, where $C(accept) = \langle \{Ann, Bob\}, \{\} \rangle$. We have $-C(accept) = \langle \{Ann, Bob, Pete\}, \{Pete\} \rangle$. Thus, $(B_I \otimes -C)(accept) = \langle \{Pete\}, \emptyset \rangle$, and $((B_I \otimes -C) \oplus C)(accept) = \langle \{Ann, Bob, Pete\}, \emptyset \rangle = B_R(accept)$.

The second revision, B'_R, is the revision of B_I by C', where $C'(accept) = \langle \{\}, \{Pete\} \rangle$. □

There is a one-to-one correspondence θ between $\mathcal{T}$-valuations (of revision atoms) and $\mathcal{T}^2$-valuations (of atoms). For a $\mathcal{T}$-valuation v, the $\mathcal{T}^2$-valuation $\theta(v)$ is defined by: $\theta(v)(a) = \langle v(\mathbf{in}(a)), v(\mathbf{out}(a)) \rangle$. The inverse mapping of θ is denoted by θ^{-1}. By using the mapping θ,

the notions of satisfaction defined earlier for $\mathcal{T}$-valuations can be extended to $\mathcal{T}^2$-valuations. Similarly, the operator T_P gives rise to a related operator T_P^b. The operator T_P^b is defined on the set of all $\mathcal{T}^2$-valuations by $T_P^b = \theta \circ T_P \circ \theta^{-1}$. The key property of the operator T_P^b is its $\leq_k$-monotonicity.

Theorem 6.1 *Let P be an annotated revision program and let B and B' be two $\mathcal{T}^2$-valuations such that $B \leq_k B'$. Then, $T_P^b(B) \leq_k T_P^b(B')$.*

By the Tarski-Knaster Theorem [31] it follows that the operator T_P^b has a least fixpoint in $\mathcal{T}^2$ (see also [13]). This fixpoint is an analogue of the concept of a least Herbrand model of a Horn program. It represents the set of annotated revision atoms that are implied by the program and, hence, must be satisfied by any revision under P of *any* initial valuation. Given an annotated revision program P we will refer to the least fixpoint of the operator T_P^b as the *necessary change* of P and will denote it by $NC(P)$. The present concept of the necessary change generalizes the corresponding notion introduced in Section 2.4 for the original unannotated revision programs.

For illustration purposes we will consider two special lattices. The first of them is the lattice with the domain $[0, 1]$ (interval of reals), with the standard ordering $\leq$, and the standard complement operation $\bar{\alpha} = 1 - \alpha$. We will denote this lattice by $\mathcal{T}_{[0,1]}$. Intuitively, the annotated revision atom $(\mathbf{in}(a){:}x)$, where $x \in [0, 1]$, stands for the statement that a is "in" with likelihood (certainty) x.

The second lattice is the Boolean algebra of all subsets of a given set X. It will be denoted by $\mathcal{T}_X$. We will think of elements from X as experts. The annotated revision atom $(\mathbf{out}(a){:}Y)$, where $Y \subseteq X$, will be understood as saying that a is believed to be "out" by those experts that are in Y (the atom $(\mathbf{in}(a){:}Y)$ has a similar meaning).

6.2 Models and s-models

The semantics of annotated revision programs will be based on the notion of a model, as defined in the previous section, and on its refinements. The first two results describe some simple properties of models of annotated revision programs. The first of them characterizes models in terms of the operator T_P^b.

Theorem 6.2 *Let P be an annotated revision program. A $\mathcal{T}^2$-valuation B is a model of P (satisfies P) if and only if $B \geq_k T_P^b(B)$.*

Models of annotated revision programs are closed under meets. This property is analogous to a similar property holding for models of Horn programs. Indeed, since $B_1 \otimes B_2 \leq_k B_i$, $i = 1, 2$, and T_P^b is $\leq_k$-monotone, by Theorem 6.2 we obtain

$$T_P^b(B_1 \otimes B_2) \leq_k T_P^b(B_i) \leq_k B_i, \;\; i = 1, 2.$$

Consequently,

$$T_P^b(B_1 \otimes B_2) \leq_k B_1 \otimes B_2.$$

Thus, again by Theorem 6.2, we obtain the following result.

Corollary 6.1 *The meet of two models of an annotated revision program P is also a model of P.*

Given an annotated revision program P, its necessary change $NC(P)$ satisfies $NC(P) = T_P^b(NC(P))$. Hence, $NC(P)$ is a model of P.

As we will now argue, not all models are appropriate for describing the meaning of an annotated revision program. The problem is that $\mathcal{T}^2$-valuations may contain inconsistent information about elements from U. When studying the meaning of an annotated revision program we will be interested in only those models whose inconsistencies are limited to those explicitly or implicitly supported by the program and by the model itself.

Consider the program $P = \{(\mathbf{in}(a){:}\{q\}) \leftarrow\}$ (where the annotation $\{q\}$ comes from the lattice $\mathcal{T}_{\{p,q\}}$). This program asserts that a is "in", according to expert q. By the closed world assumption, it also implies an upper bound for the evidence for $\mathbf{out}(a)$. In this case the only expert that might possibly believe in $\mathbf{out}(a)$ is p (this is to say that expert q does not believe in $\mathbf{out}(a)$). Observe that a $\mathcal{T}^2$-valuation B, such that $B(a) = \langle\{q\}, \{q\}\rangle$ is a model of P but it does not satisfy the implicit bound on evidence for $\mathbf{out}(a)$.

Let P be an annotated program and let B be a $\mathcal{T}^2$-valuation that is a model of P. By the explicit evidence we mean evidence provided by heads of program rules applicable with respect to B, that is with bodies satisfied by B. It is $T_P^b(B)$. The implicit information is given by a version of the closed world assumption: if the maximum evidence for a revision atom l provided by the program is α, then the evidence for the dual revision atom l^D ($\mathbf{out}(a)$, if $l = \mathbf{in}(a)$, or $\mathbf{in}(a)$, otherwise) must not exceed $\bar{\alpha}$ (unless explicitly forced by the program). Thus, the implicit evidence is given by $-T_P^b(B)$. Hence, a model B of a program P contains no more evidence than what is directly

implied by P given B and what is indirectly implied by P given B if $B \leq_k T_P^b(B) \oplus (-T_P^b(B))$ (since the direct evidence is given by $T_P^b(B)$ and the implicit evidence is given by $-T_P^b(B)$). This observation leads us to a refinement of the notion of a model of an annotated revision program.

Definition 6.1 *Let P be an annotated revision program and let B be a $\mathcal{T}^2$-valuation. We say that B is an* s-model *of P if*

$$T_P^b(B) \leq_k B \leq_k T_P^b(B) \oplus (-T_P^b(B)).$$

△

The "s" in the term "s-model" stands for "supported" and emphasizes that inconsistencies in s-models are limited to those explicitly or implicitly supported by the program and the model itself.

Clearly, by Theorem 6.2, an s-model of P is a model of P. In addition, it is easy to see that the necessary change of an annotated program P is an s-model of P (it follows directly from the fact that $NC(P) = T_P^b(NC(P))$).

The distinction between models and s-models appears only in the context of inconsistent information. This observation is formally stated below.

Theorem 6.3 *Let P be an annotated revision program. A consistent $\mathcal{T}^2$-valuation B is an s-model of P if and only if B is a model of P.*

Proof.
($\Rightarrow$) Let B be an s-model of P. Then, $T_P^b(B) \leq_k B \leq_k T_P^b(B) \oplus (-T_P^b(B))$. In particular, $T_P^b(B) \leq_k B$ and, by Theorem 6.2, B is a model of P.
($\Leftarrow$) Let B satisfy P. From Theorem 6.2 we have $T_P^b(B) \leq_k B$. Hence, $-B \leq_k -T_P^b(B)$. Since B is consistent, $B \leq_k -B$. Therefore,

$$T_P^b(B) \leq_k B \leq_k -B \leq_k -T_P^b(B). \tag{6.1}$$

It follows that $T_P^b(B) \leq_k -T_P^b(B)$ and $T_P^b(B) \oplus (-T_P^b(B)) = -T_P^b(B)$. By (6.1), we get

$$T_P^b(B) \leq_k B \leq_k T_P^b(B) \oplus (-T_P^b(B))$$

and the assertion follows. □

Some of the properties of ordinary models hold for s-models, too. For instance, the following theorem shows that an s-model of two annotated revision programs is an s-model of their union.

Theorem 6.4 *Let P_1, P_2 be annotated revision programs. Let B be an s-model of P_1 and an s-model of P_2. Then, B is an s-model of $P_1 \cup P_2$.*

Proof.
Clearly, B is a model of $P_1 \cup P_2$. That is,

$$T^b_{P_1 \cup P_2}(B) \leq_k B. \tag{6.2}$$

It is easy to see that $T^b_{P_1 \cup P_2}(B) = T^b_{P_1}(B) \oplus T^b_{P_2}(B)$. Hence, by the de Morgan law,

$$-T^b_{P_1 \cup P_2}(B) = -T^b_{P_1}(B) \otimes -T^b_{P_2}(B).$$

By the definition of an s-model:

$$T^b_{P_1}(B) \leq_k B \leq_k T^b_{P_1}(B) \oplus -T^b_{P_1}(B), \text{ and}$$

$$T^b_{P_2}(B) \leq_k B \leq_k T^b_{P_2}(B) \oplus -T^b_{P_2}(B).$$

Therefore, by the distributivity of lattice operations in $\mathcal{T}^2$,

$$B \leq_k (T^b_{P_1}(B) \oplus -T^b_{P_1}(B)) \otimes (T^b_{P_2}(B) \oplus -T^b_{P_2}(B)) =$$

$$= (T^b_{P_1}(B) \otimes (T^b_{P_2}(B) \oplus -T^b_{P_2}(B))) \oplus (-T^b_{P_1}(B) \otimes (T^b_{P_2}(B) \oplus -T^b_{P_2}(B))) \leq_k$$

$$\leq_k T^b_{P_1}(B) \oplus (-T^b_{P_1}(B) \otimes (T^b_{P_2}(B) \oplus -T^b_{P_2}(B))) =$$

$$= T^b_{P_1}(B) \oplus (-T^b_{P_1}(B) \otimes T^b_{P_2}(B)) \oplus (-T^b_{P_1}(B) \otimes -T^b_{P_2}(B)) \leq_k$$

$$\leq_k T^b_{P_1}(B) \oplus T^b_{P_2}(B) \oplus -T^b_{P_1 \cup P_2}(B) = T^b_{P_1 \cup P_2}(B) \oplus -T^b_{P_1 \cup P_2}(B).$$

In other words,

$$B \leq_k T^b_{P_1 \cup P_2}(B) \oplus -T^b_{P_1 \cup P_2}(B). \tag{6.3}$$

From (6.2) and (6.3) it follows that B is an s-model of $P_1 \cup P_2$. □

Not all of the properties of models hold for s-models. For instance, the counterpart of Corollary 6.1 does not hold. The following example shows that the meet of two s-models is not necessarily an s-model.

Example 6.4 Consider the lattice $\mathcal{T}_{\{p,q\}}$. Let P be an annotated program consisting of the following rules:

$$\begin{aligned}(\mathbf{in}(a){:}\{p\}) &\leftarrow (\mathbf{in}(b){:}\{p\})\\ (\mathbf{out}(a){:}\{p\}) &\leftarrow \\ (\mathbf{in}(a){:}\{p\}) &\leftarrow (\mathbf{out}(b){:}\{p\})\end{aligned}$$

Let B_1 and B_2 be defined as follows.

$$B_1(a) = \langle\{p\},\{p\}\rangle, \qquad B_1(b) = \langle\{p\},\emptyset\rangle;$$

$$B_2(a) = \langle\{p\},\{p\}\rangle, \qquad B_2(b) = \langle\emptyset,\{p\}\rangle.$$

Let us show that B_1 is an s-model of P. Indeed,

$$T_P^b(B_1)(a) = \langle\{p\},\{p\}\rangle, \qquad T_P^b(B_1)(b) = \langle\emptyset,\emptyset\rangle.$$

Hence,

$$-T_P^b(B_1)(a) = \langle\{q\},\{q\}\rangle, \qquad -T_P^b(B_1)(b) = \langle\{p,q\},\{p,q\}\rangle.$$

Therefore,

$$T_P^b(B_1)(a) \leq_k B_1(a) \leq_k (T_P^b(B_1) \oplus -T_P^b(B_1))(a), \text{ and}$$

$$T_P^b(B_1)(b) \leq_k B_1(b) \leq_k (T_P^b(B_1) \oplus -T_P^b(B_1))(b).$$

In other words, B_1 is an s-model of P. Similarly, B_2 is an s-model of P. However, $B_1 \otimes B_2$ is *not* an s-model of P. Indeed,

$$(B_1 \otimes B_2)(a) = \langle\{p\},\{p\}\rangle, \qquad (B_1 \otimes B_2)(b) = \langle\emptyset,\emptyset\rangle.$$

Then,

$$T_P^b(B_1 \otimes B_2)(a) = \langle\emptyset,\{p\}\rangle, \qquad T_P^b(B_1 \otimes B_2)(b) = \langle\emptyset,\emptyset\rangle, \text{ and}$$

$$-T_P^b(B_1 \otimes B_2)(a) = \langle\{q\},\{p,q\}\rangle, \qquad -T_P^b(B_1 \otimes B_2)(b) = \langle\{p,q\},\{p,q\}\rangle.$$

Hence,

$$(B_1 \otimes B_2)(a) \not\leq_k (T_P^b(B_1 \otimes B_2) \oplus -T_P^b(B_1 \otimes B_2))(a) = \langle\{q\},\{p,q\}\rangle.$$

Therefore, $B_1 \otimes B_2$ is not an s-model of P. □

In this example both B_1 and B_2, as well as their meet $B_1 \otimes B_2$ are inconsistent. For B_1 and B_2 there are rules in P that explicitly imply their inconsistencies. However, for $B_1 \otimes B_2$ the bodies of these rules are no longer satisfied. Consequently, the inconsistency in $B_1 \otimes B_2$ is not implied by P. That is, $B_1 \otimes B_2$ is not an s-model of P.

Let us now investigate what happens when we add to an annotated revision program P a rule $r = (l{:}\alpha) \leftarrow (l{:}\alpha)$ (here l is a revision atom, α is an annotation). Unlike ordinary revision programs where every database is a model of a rule of the form $l \leftarrow l$, not every $\mathcal{T}^2$-valuation is an s-model of r. Therefore, adding such a rule may affect the set of s-models of the program. On the one hand, rule r by imposing an additional implicit bound on l^D may give rise to a situation when an s-model of P is not an s-model of $P \cup \{r\}$ (Case 1 of Example 6.5). On the other hand, rule r may provide additional explicit evidence for l that results in a situation when an s-model of $P \cup \{r\}$ is not an s-model of P (Case 2 of Example 6.5).

Example 6.5 Let $U = \{a\}$ and the lattice of annotations be $\mathcal{T}_{\{p,q\}}$. Let $B(a) = \langle\{p\}, \{p\}\rangle$. Let $r = (\mathbf{in}(a){:}\{p\}) \leftarrow (\mathbf{in}(a){:}\{p\})$.

1. Let $P = \{\}$. Then, $T_P^b(B)(a) = \langle\emptyset, \emptyset\rangle$, and $-T_P^b(B)(a) = \langle\{p,q\}, \{p,q\}\rangle$. Hence, $T_P^b(B)(a) \leq_k B(a) \leq_k T_P^b(B)(a) \oplus (-T_P^b(B))(a)$. Thus, B is an s-model of P. However, B is *not* an s-model of $P \cup \{r\}$. Indeed, $T_{P\cup\{r\}}^b(B)(a) = \langle\{p\}, \emptyset\rangle$, and $-T_{P\cup\{r\}}^b(B)(a) = \langle\{p,q\}, \{q\}\rangle$. Hence, $B(a) \not\leq_k T_{P\cup\{r\}}^b(B)(a) \oplus (-T_{P\cup\{r\}}^b(B))(a) = \langle\{p,q\}, \{q\}\rangle$. Therefore, B is *not* an s-model of $P \cup \{r\}$.

2. Let $P = \{\ (\mathbf{out}(a){:}\{p\}) \leftarrow\ \}$. Then it is easy to see that B is *not* an s-model of P. However, B is an s-model of $P \cup \{r\}$. □

Remark 6.3 *Let us note that adding rule $r = (l:\alpha) \leftarrow (l:\alpha)$ to P has no effect on consistent models of P. Indeed, let B be a consistent model of P. Clearly, B is a model of $\{r\}$. Hence, by Theorem 6.3, B is an s-model of P, and an s-model of $\{r\}$. Therefore, Theorem 6.4 implies that B is an s-model of $P \cup \{r\}$.*

6.3 Justified revisions of annotated revision programs.

6.3.1 Definition.

In this section, we will extend to the case of annotated revision programs the notion of a justified revision introduced for revision programs (Section 2.4).

There are several properties that one would expect to hold when the notion of justified revision is extended to the case of programs with annotations. Clearly, the extended concept should specialize to the original definition if annotations are dropped. Next, main properties of justified revisions studied in [23, 18] should have their counterparts in the case of justified revisions of annotated programs. In particular, justified revisions of an annotated revision program should be models of the program.

There is one other requirement that naturally arises in the context of programs with annotations. Consider two annotated revision rules r and r' that are exactly the same except that the body of r contains two annotated revision atoms $(l{:}\beta_1)$ and $(l{:}\beta_2)$, while the body of r' instead of $(l{:}\beta_1)$ and $(l{:}\beta_2)$ contains annotated revision atom $(l{:}\beta_1 \vee \beta_2)$.

$$r = \quad \ldots \leftarrow \ldots, (l{:}\beta_1), \ldots, (l{:}\beta_2), \ldots$$

$$r' = \quad \ldots \leftarrow \ldots, (l{:}\beta_1 \vee \beta_2), \ldots$$

We will refer to this operation as the *join transformation*.

It is clear, that a $\mathcal{T}^2$-valuation B satisfies $(l{:}\beta_1)$ and $(l{:}\beta_2)$ if and only if B satisfies $(l{:}\beta_1 \vee \beta_2)$. Consequently, replacing rule r by rule r' (or vice versa) in an annotated revision program should have no effect on justified revisions. In fact, any reasonable semantics for annotated revision programs should be invariant under such operation, and we will refer to this property of a semantics of annotated revision programs as *invariance under join*.

Now we introduce the notion of a justified revision of an annotated revision program and contrast it with an earlier proposal by Fitting [9]. In the following section we show that our concept of a justified revision satisfies all the requirements listed above.

Let a $\mathcal{T}^2$-valuation B_I represent our current knowledge about some subset of the universe U. Let an annotated revision program P describe an update that B_I should be subject to. The goal is to identify a class of $\mathcal{T}^2$-valuations that could be viewed as representing updated information about

the subset, obtained by revising B_I by P. As argued in [22, 23], each appropriately "revised" valuation B_R must be *grounded* in P and in B_I, that is, any difference between B_I and the revised $\mathcal{T}^2$-valuation B_R must be justified by means of the program and the information available in B_I.

To determine whether B_R is grounded in B_I and P, we use the *reduct* of P with respect to these two valuations. The construction of the reduct consists of two steps and mirrors the original definition of the reduct of an unannotated revision program [23]. In the first step, we eliminate from P all rules whose bodies are not satisfied by B_R (their use does not have an *a posteriori* justification with respect to B_R). In the second step, we take into account the initial valuation B_I.

How can we use the information about the initial $\mathcal{T}^2$-valuation B_I at this stage? Assume that B_I provides evidence α for a revision atom l. Assume also that an annotated revision atom $(l{:}\beta)$ appears in the body of a rule r. In order to satisfy this premise of the rule, it is enough to derive, from the program resulting from step 1, an annotated revision atom $(l{:}\gamma)$, where $\alpha \vee \gamma \geq \beta$. The least such element exists (due to the fact that $\mathcal{T}$ is complete and infinitely distributive). Let us denote this value by *pcomp*(α, β). (The operation *pcomp*$(\cdot, \cdot)$ is known in the lattice theory as the *relative pseudocomplement*, see [27].)

Thus, in order to incorporate information about a revision atom l contained in the initial $\mathcal{T}^2$-valuation B_I, which is given by $\alpha = (\theta^{-1}(B_I))(l)$, we proceed as follows. In the bodies of rules of the program obtained after step 1, we replace each annotated revision atom of the form $(l{:}\beta)$ by the annotated revision atom $(l{:}$*pcomp*$(\alpha, \beta))$.

Now we are ready to formally introduce the notion of *reduct* of an annotated revision program P with respect to the pair of $\mathcal{T}^2$-valuations, the initial one, B_I, and a candidate for a revised one, B_R.

Definition 6.2 *The* reduct $P_{B_R}|B_I$ *is obtained from* P *by*

1. *removing every rule whose body contains an annotated atom that is not satisfied in* B_R,
2. *replacing each annotated atom* $(l{:}\beta)$ *from the body of each remaining rule by the annotated atom* $(l{:}\gamma)$, *where* $\gamma =$ *pcomp*$((\theta^{-1}(B_I))(l), \beta)$. $\triangle$

We now define the concept of a *justified revision*. Given an annotated revision program P, we first compute the reduct $P_{B_R}|B_I$ of the program P with respect to B_I and B_R. Next, we compute

the necessary change for the reduced program. Finally we apply this change to the $\mathcal{T}^2$-valuation B_I. A $\mathcal{T}^2$-valuation B_R is a justified revision of B_I if the result of these three steps is B_R. Thus we have the following definition.

Definition 6.3 *B_R is a* P-justified revision *of B_I if $B_R = (B_I \otimes -C) \oplus C$, where $C = NC(P_{B_R}|B_I)$ is the necessary change for $P_{B_R}|B_I$.* △

We will now contrast this approach with the one proposed by Fitting in [9]. In order to do so, we recall the definitions introduced in [9]. The key difference is in the way Fitting defines the reduct of a program. The first step is the same in both approaches. However, the second steps, in which the initial valuation is used to simplify the bodies of the rules not eliminated in the first step of the construction, differ.

Definition 6.4 (Fitting) *Let P be an annotated revision program and let B_I and B_R be $\mathcal{T}^2$-valuations. The* F-reduct *of P with respect to (B_I, B_R) (denoted $P^F_{B_R}|B_I$) is defined as follows:*

1. *Remove from P every rule whose body contains an annotated revision atom that is not satisfied in B_R.*

2. *From the body of each remaining rule delete any annotated revision atom that is satisfied in B_I.* △

The notion of justified revision as defined by Fitting differs from our notion only in that it uses the necessary change of the F-reduct (instead of the necessary change of the reduct defined above in Definition 6.2). We call the justified revision based on the notion of F-reduct, the *F-justified revision*.

In the remainder of this section we show that the notion of the F-justified revision does not in general satisfy some basic requirements that we would like justified revisions to have. In particular, F-justified revisions under an annotated revision program P are not always models of P.

Example 6.6 Consider the lattice $\mathcal{T}_{\{p,q\}}$. Let P be a program consisting of the following rules:

$$(\mathbf{in}(a){:}\{p\}) \leftarrow (\mathbf{in}(b){:}\{p,q\}) \quad \text{and} \quad (\mathbf{in}(b){:}\{q\}) \leftarrow$$

and let B_I be a valuation such that $B_I(a) = \langle \emptyset, \emptyset \rangle$ and $B_I(b) = \langle \{p\}, \emptyset \rangle$. Let B_R be a valuation given by $B_R(a) = \langle \emptyset, \emptyset \rangle$ and $B_R(b) = \langle \{p, q\}, \emptyset \rangle$. Clearly, $P^F_{B_R} | B_I = P$, and B_R is an F-justified revision of B_I (under P). However, B_R does not satisfy P. □

The semantics of F-justified revisions also fails to satisfy the invariance under join property.

Example 6.7 Let P be the same revision program as before, and let P' consist of the rules

$$(\mathbf{in}(a){:}\{p\}) \leftarrow (\mathbf{in}(b){:}\{p\}), (\mathbf{in}(b){:}\{q\}) \quad \text{and} \quad (\mathbf{in}(b){:}\{q\}) \leftarrow$$

Let the initial valuation B_I be given by $B_I(a) = \langle \emptyset, \emptyset \rangle$ and $B_I(b) = \langle \{p\}, \emptyset \rangle$. The only F-justified revision of B_I (under P) is a $\mathcal{T}^2$-valuation B_R, where $B_R(a) = \langle \emptyset, \emptyset \rangle$ and $B_R(b) = \langle \{p, q\}, \emptyset \rangle$. The only F-justified revision of B_I (under P') is a $\mathcal{T}^2$-valuation B'_R, where $B'_R(a) = \langle \{p\}, \emptyset \rangle$ and $B'_R(b) = \langle \{p, q\}, \emptyset \rangle$. Thus, replacing in the body of a rule $(\mathbf{in}(b){:}\{p, q\})$ by $(\mathbf{in}(b){:}\{p\})$ and $(\mathbf{in}(b){:}\{q\})$ affects F-justified revisions. □

However, in some cases the two definitions of justified revision coincide. The following theorem provides a complete characterization of those cases (let us recall that a lattice $\mathcal{T}$ is *linear* if for any two elements $\alpha, \beta \in \mathcal{T}$ either $\alpha \leq \beta$ or $\beta \leq \alpha$).

Theorem 6.5 *F-justified revisions and justified revisions coincide if and only if the lattice $\mathcal{T}$ is linear.*

Proof.
($\Rightarrow$) Assume that F-justified revisions and justified revisions coincide for a lattice $\mathcal{T}$. Let $\alpha, \beta \in \mathcal{T}$. We will show that either $\alpha \leq \beta$ or $\beta \leq \alpha$. Indeed, let P be annotated revision program consisting of the following rules.

$$(\mathbf{in}(a){:}\alpha) \leftarrow (\mathbf{in}(b){:}\alpha \vee \beta) \quad \text{and} \quad (\mathbf{in}(b){:}\beta) \leftarrow$$

Let B_I be given by $B_I(a) = \langle \bot, \bot \rangle$ and $B_I(b) = \langle \alpha, \bot \rangle$. Let B_R be given by $B_R(a) = \langle \alpha, \bot \rangle$ and $B_R(b) = \langle \alpha \vee \beta, \bot \rangle$. It is easy to see that B_R is a justified revision of B_I (with respect to P). By our assumption, B_R is also an F-justified revision of B_I. There are only two possible cases.
Case 1. $\alpha \vee \beta \leq \alpha$. Then, $\beta \leq \alpha$.

Case 2. $\alpha \vee \beta \not\leq \alpha$. Then, $P^F_{B_R}|B_I = P$. Let $C = NC(P^F_{B_R}|B_I)$. By the definition of the necessary change,

$$C(a) = NC(P^F_{B_R}|B_I)(a) = NC(P)(a) = \begin{cases} \langle \bot, \bot \rangle, & \text{when } \alpha \vee \beta \not\leq \beta \\ \langle \alpha, \bot \rangle, & \text{when } \alpha \vee \beta \leq \beta. \end{cases}$$

By the definition of an F-justified revision, $B_R = (B_I \otimes -C) \oplus C$. From the facts that $B_R(a) = \langle \alpha, \bot \rangle$ and $B_I(a) = \langle \bot, \bot \rangle$ it follows that $C(a) = \langle \alpha, \bot \rangle$. Therefore, it is the case that $\alpha \vee \beta \leq \beta$. That is, $\alpha \leq \beta$.

($\Leftarrow$) Assume that lattice $\mathcal{T}$ is linear. Then, for any $\alpha, \beta \in \mathcal{T}$,

$$\mathit{pcomp}(\alpha, \beta) = \begin{cases} \bot, & \text{when } \alpha \geq \beta \\ \beta & \text{otherwise (when } \alpha < \beta). \end{cases}$$

Let P be an annotated revision program. Let B_I and B_R be any $\mathcal{T}^2$-valuations. Let us see what is the difference between $P_{B_R}|B_I$ and $P^F_{B_R}|B_I$. The first steps in the definitions of reduct and F-reduct are the same. During the second step of the definition of an F-reduct each annotated atom $(l{:}\beta)$ such that $\beta \leq B_I(l)$ is deleted from bodies of rules. In the second step of the definition of the reduct such annotated atom is replaced by $(l{:}\bot)$. If $\beta > B_I(l)$, then in the reduct $P_{B_R}|B_I$ annotated atom $(l{:}\beta)$ is replaced by $(l{:}\mathit{pcomp}(B_I(l), \beta)) = (l{:}\beta)$, that is, it remains as it is. In the F-reduct, $(l:\beta)$ also remains in the bodies for $\beta > B_I(l)$. Thus, the only difference between $P_{B_R}|B_I$ and $P^F_{B_R}|B_I$ is that bodies of the rules from $P_{B_R}|B_I$ may contain atoms of the form $(l : \bot)$, where $l \in U$, that are not present in the bodies of the corresponding rules in $P^F_{B_R}|B_I$. However, annotated atoms of the form $(l : \bot)$ are always satisfied. Therefore, the necessary changes of $P_{B_R}|B_I$ and $P^F_{B_R}|B_I$, as well as justified and F-justified revisions of B_I coincide. □

Theorem 6.5 explains why the difference between the justified revisions and F-justified revisions is not seen when we limit our attention to revision programs as considered in [23]. Namely, the lattice $\mathcal{TWO} = \{\mathbf{f}, \mathbf{t}\}$ of boolean values is linear. Similarly, the lattice of reals from the interval $[0, 1]$ is linear, and there the differences cannot be seen either.

6.3.2 Properties.

In this section we study basic properties of justified revisions. We show that key properties of justified revisions in the case of revision programs without annotations have their counterparts in the case of justified revisions of annotated revision programs.

First, we observe that ordinary revision programs as defined in Section 2.4 can be encoded as annotated revision programs (with annotations taken from the lattice $\mathcal{TWO} = \{\mathbf{f}, \mathbf{t}\}$). Namely, a revision rule

$$p \leftarrow q_1, \ldots q_m$$

(where p and all q_i's are revision atoms) can be encoded as

$$(p{:}\mathbf{t}) \leftarrow (q_1{:}\mathbf{t}), \ldots, (q_m{:}\mathbf{t})$$

We will denote by P^a the result of applying this transformation to a revision program P (rule by rule). Second, let us represent a set of atoms B by a $\mathcal{TWO}^2$-valuation B^v as follows: $B^v(a) = \langle \mathbf{t}, \mathbf{f} \rangle$, if $a \in B$, and $B^v(a) = \langle \mathbf{f}, \mathbf{t} \rangle$, otherwise.

Fitting [9] argued that under such encodings the semantics of F-justified revisions generalizes the semantics of justified revisions introduced in [22]. Since for lattices whose ordering is linear the approach by Fitting and the approach presented in Section 6.3.1 coincide, and since the ordering of $\mathcal{TWO}$ is linear, the semantics of justified revisions discussed here extends the semantics of justified revisions from [22]. Specifically, we have the following result.

Theorem 6.6 *Let P be an ordinary revision program and let B_I and B_R be two sets of atoms. Then, B_R is a P-justified revision of B_I if and only if the necessary change of $P^a_{B^v_R} | B^v_I$ is consistent and B^v_R is a P^a-justified revision of B^v_I.*

Before we study how properties of justified revisions generalize to the case with annotations, we prove the following auxiliary results.

Lemma 6.1 *Let P be an annotated revision program. Let B be a $\mathcal{T}^2$-valuation. Then $NC(P_B|B) = T^b_P(B)$.*

Proof.
The assertion follows from definitions of a necessary change and operator T^b_P. □

Lemma 6.2 *Let P be an annotated revision program. Let B_I, B_R, and C be $\mathcal{T}^2$-valuations, such that $B_R \leq B_I \oplus C$. Then, C satisfies the bodies of all rules in $P_{B_R}|B_I$.*

Proof.

Let $r' \in P_{B_R}|B_I$. Let $(l{:}\gamma)$ be an annotated revision atom from the body of r'. Let $(\theta^{-1}(B_I))(l) = \alpha$. By the definition of the reduct, r' was obtained from some rule $r \in P$, such that the body of r is satisfied by B_R, and $\gamma = \mathit{pcomp}(\alpha, \beta)$, where $(l{:}\beta)$ is in the body of r. Since the body of r is satisfied by B_R, we have $\beta \leq (\theta^{-1}(B_R))(l)$. From $B_R \leq_k B_I \oplus C$ it follows that

$$(\theta^{-1}(B_R))(l) \leq (\theta^{-1}(B_I \oplus C))(l) =$$

$$= (\theta^{-1}(B_I))(l) \vee (\theta^{-1}(C))(l) = \alpha \vee (\theta^{-1}(C))(l).$$

Combining this inequality with our previous observation that $\beta \leq (\theta^{-1}(B_R))(l)$, we get $\beta \leq \alpha \vee (\theta^{-1}(C))(l)$. By the definition of $\mathit{pcomp}(\alpha, \beta)$, we get $\gamma \leq (\theta^{-1}(C))(l)$. That is, C satisfies $(l : \gamma)$. Since $(l : \gamma)$ was arbitrary, C satisfies all annotated revision atoms in the body of r'. As r' was an arbitrary rule from $P_{B_R}|B_I$, we conclude that C satisfies the bodies of all rules in $P_{B_R}|B_I$. □

Lemma 6.3 *Let B_R be a P-justified revision of B_I. Then, $NC(P_{B_R}|B_I) = T_P^b(B_R)$.*

Proof.

By the definition of a justified revision $B_R = (B_I \otimes -C) \oplus C$, where $C = NC(P_{B_R}|B_I)$. Hence, $B_R \leq B_I \oplus C$. By Lemma 6.2, C satisfies the bodies of all rules in $P_{B_R}|B_I$. Since C is a model of $P_{B_R}|B_I$, C satisfies all heads of clauses in $P_{B_R}|B_I$.

Let D be a valuation satisfying all heads of rules in $P_{B_R}|B_I$. Then D is a model of $P_{B_R}|B_I$. Since C is the least model of the reduct $P_{B_R}|B_I$, we find that $C \leq_k D$. Consequently, C is the least valuation that satisfies all heads of the rules in $P_{B_R}|B_I$. The rules in P_{B_R} are all those rules from P whose bodies are satisfied by B_R. Thus, by the definition of the operator T_P^b, $C = T_P^b(B_R)$. □

We will now look at properties of the semantics of justified revisions. We will present a series of results generalizing properties of revision programs to the case with annotations. We will show that the concept of an s-model is a useful notion in the investigations of justified revisions of annotated programs.

Our first result relates justified revisions to models and s-models. Let us recall that in the case of revision programs without annotations, justified revisions under a revision program P are models of P. In the case of annotated revision programs we have an analogous result.

Theorem 6.7 *Let P be an annotated revision program and let B_I and B_R be $\mathcal{T}^2$-valuations. If B_R is a P-justified revision of B_I then B_R is an s-model of P (and, hence, a model of P).*

Proof.
By the definition of a P-justified revision, $B_R = (B_I \otimes -C) \oplus C$, where C is the necessary change for $P_{B_R}|B_I$. From Lemma 6.3 it follows that $C = T_P^b(B_R)$. Therefore,

$$B_R = (B_I \otimes -T_P^b(B_R)) \oplus T_P^b(B_R) \leq_k -T_P^b(B_R) \oplus T_P^b(B_R).$$

Also,

$$B_R = (B_I \otimes -T_P^b(B_R)) \oplus T_P^b(B_R) \geq T_P^b(B_R).$$

Hence, B_R is an s-model of P. □

In the previous section we showed an example demonstrating that F-justified revisions do not satisfy the property of invariance under joins. In contrast, justified revisions in the sense of Section 6.3.1 do have this property.

Theorem 6.8 *Let P_2 be the result of simplification of an annotated revision program P_1 by means of the join transformation. Then for every initial database B_I, P_1-justified revisions of B_I coincide with P_2-justified revisions of B_I.*

The proof follows directly from the definition of P-justified revisions and from the following distributivity property of pseudocomplement: $\mathit{pcomp}(\alpha, \beta_1) \vee \mathit{pcomp}(\alpha, \beta_2) = \mathit{pcomp}(\alpha, \beta_1 \vee \beta_2)$.

In the case of revision programs without annotations, a model of a program P is its own unique P-justified revision (Theorem 2.11). In the case of programs with annotations, the situation is slightly more complicated. The next several results provide a complete description of justified revisions of models of annotated revision programs. First, we characterize those models that are their own justified revisions. This result provides additional support for the importance of the notion of an s-model in the study of annotated revision programs.

Theorem 6.9 *Let a $\mathcal{T}^2$-valuation B_I be a model of an annotated revision program P. Then, B_I is a P-justified revision of itself if and only if B_I is an s-model of P.*

Proof.

Let us denote $C = NC(P_{B_I}|B_I)$. By the definition, B_I is a P-justified revision of itself if and only if $B_I = (B_I \otimes -C) \oplus C$. Since B_I satisfies P, Theorem 6.2 implies that $B_I \geq_k C$. Thus, $B_I \oplus C = B_I$. Distributivity of the product lattice $\mathcal{T}^2$ implies that $(B_I \otimes -C) \oplus C = (B_I \oplus C) \otimes (-C \oplus C) = B_I \otimes (-C \oplus C)$. Clearly, $B_I = B_I \otimes (-C \oplus C)$ if and only if $B_I \leq_k (-C \oplus C)$.

By Lemma 6.1, $C = NC(P_{B_I}|B_I) = T_P^b(B_I)$. Thus, B_I is a P-justified revision of itself if and only if $B_I \leq_k T_P^b(B_I) \oplus (-T_P^b(B_I))$. But this latter condition is precisely the one that distinguishes s-models among models. Thus, under the assumptions of the theorem, B_I is a P-justified revision of itself if and only if it is an s-model of P. □

As we observed above, in the case of programs without annotations, models of a revision program are their own *unique* justified revisions. This property does not hold, in general, in the case of annotated revision programs. In other words, s-models, if they are inconsistent, may have other revisions besides themselves (by Theorem 6.9 they always are their own revisions).

The following example shows that an inconsistent s-model may have no revisions other than itself, may have only one consistent justified revision, or may have incomparable (with respect to the knowledge ordering) consistent revisions.

Example 6.8 Let the lattice of annotations be $\mathcal{T}_{\{p,q\}}$. Consider an inconsistent $\mathcal{T}^2$-valuation B_I such that $B_I(a) = \langle \{q\}, \{q\} \rangle$.

1. Consider annotated revision program P_1 consisting of the clauses:

$$(\mathbf{out}(a){:}\{q\}) \leftarrow \quad \text{and} \quad (\mathbf{in}(a){:}\{q\}) \leftarrow$$

   It is easy to see that B_I is an s-model of P_1 and the only justified revision of itself.

2. Let an annotated revision program P_2 consist of the clauses:

$$(\mathbf{out}(a){:}\{q\}) \leftarrow \quad \text{and} \quad (\mathbf{in}(a){:}\{q\}) \leftarrow (\mathbf{in}(a){:}\{q\})$$

   Then B_I is an s-model of P_2. Hence, B_I is its own justified revision (under P_2).

   However, B_I is not the only P_2-justified revision of B_I. Consider the $\mathcal{T}^2$-valuation B_R such that $B_R(a) = \langle \emptyset, \{q\} \rangle$. We have $P_{2\,B_R}|B_I = \{(\mathbf{out}(a) : \{q\}) \leftarrow\}$. Let us denote the

corresponding necessary change, $NC(P_{2\,B_R}|B_I)$, by C. Then, $C(a) = \langle \emptyset, \{q\}\rangle$. Hence, $-C = \langle \{p\}, \{p, q\}\rangle$ and $((B_I \otimes -C) \oplus C)(a) = \langle \emptyset, \{q\}\rangle = B_R(a)$. Consequently, B_R is a P_2-justified revision of B_I. It is the only consistent P_2-justified revision of B_I.

3. Let an annotated revision program P_3 be the following:

$$(\mathbf{in}(a){:}\{q\}) \leftarrow (\mathbf{in}(a){:}\{q\}) \quad \text{and} \quad (\mathbf{out}(a){:}\{q\}) \leftarrow (\mathbf{out}(a){:}\{q\})$$

Then B_I is an s-model of P_3 and its own P_3-justified revision. In addition, it is straitforward to check that B_I has two consistent revisions B_R and B'_R, where $B_R(a) = \langle \emptyset, \{q\}\rangle$ and $B'_R(a) = \langle \{q\}, \emptyset\rangle$. The revisions B_R and B'_R are incomparable with respect to the knowledge ordering. □

The same behavior can be observed in the case of programs annotated with elements from other lattices. The following example is analogous to the second case in the Example 6.8, but the lattice is $\mathcal{T}_{[0,1]}$.

Example 6.9 Let P be an annotated revision program (annotations belong to the lattice $\mathcal{T}_{[0,1]}$) consisting of the rules:

$$(\mathbf{out}(a){:}1) \leftarrow \quad \text{and} \quad (\mathbf{in}(a){:}0.4) \leftarrow (\mathbf{in}(a){:}0.4)$$

Let B_I be a valuation such that $B_I(a) = \langle 0.4, 1\rangle$. Then, B_I is an s-model of P and hence, it is its own P-justified revision. Consider a valuation B_R such that $B_R(a) = \langle 0, 1\rangle$. We have $P_{B_R}|B_I = \{(\mathbf{out}(a) : 1) \leftarrow\}$. Let us denote the necessary change $NC(P_{B_R}|B_I)$ by C. Then $C(a) = \langle 0, 1\rangle$ and $-C = \langle 0, 1\rangle$. Thus, $((B_I \otimes -C) \oplus C)(a) = \langle 0, 1\rangle = B_R(a)$. That is, B_R is a P-justified revision of B_I. □

Note that in both examples the additional justified revision B_R of B_I is smaller than B_I with respect to the ordering $\leq_k$. It is not coincidental as demonstrated by the next result.

Theorem 6.10 *Let B_I be a model of an annotated revision program P. Let B_R be a P-justified revision of B_I. Then, $B_R \leq_k B_I$.*

Proof.

By the definition of a P-justified revision, $B_R = (B_I \otimes -C) \oplus C$, where C is the necessary change of $P_{B_R}|B_I$. By the definition of the reduct $P_{B_R}|B_I$ and the fact that B_I is a model of P, it follows that B_I is a model of $P_{B_R}|B_I$. The necessary change C is the least fixpoint of $T^b_{P_{B_R}|B_I}$, therefore, $C \leq B_I$. Hence,

$$B_R = (B_I \otimes -C) \oplus C \leq_k B_I \oplus C \leq_k B_I \oplus B_I = B_I. \qquad \square$$

Finally, we observe that if a *consistent* $\mathcal{T}^2$-valuation is a model (or an s-model; these notions coincide in the class of consistent valuations) of a program then it is its own *unique* justified revision.

Theorem 6.11 *Let B_I be a consistent model of an annotated revision program P. Then, B_I is the only P-justified revision of itself.*

Proof.

Theorem 6.3 implies that B_I is an s-model of P. Then, from Theorem 6.9 we get that B_I is a P-justified revision of itself. We need to show that there are no other P-justified revisions of B_I.

Let B_R be a P-justified revision of B_I. Then, $B_R \leq_k B_I$ (Theorem 6.10). Therefore, $T^b_P(B_R) \leq_k T^b_P(B_I)$. Hence, $-T^b_P(B_I) \leq_k -T^b_P(B_R)$. Theorem 6.2 implies that $B_I \geq_k T^b_P(B_I)$. Thus, $-B_I \leq_k -T^b_P(B_I)$. Since B_I is consistent, $B_I \leq_k -B_I$. Combining the above inequalities, we get

$$B_I \leq_k -B_I \leq_k -T^b_P(B_I) \leq_k -T^b_P(B_R).$$

That is, $B_I \leq_k -T^b_P(B_R)$. Hence, $B_I \otimes -T^b_P(B_R) = B_I$.

From definition of justified revision and Lemma 6.3,

$$B_R = (B_I \otimes -T^b_P(B_R)) \oplus T^b_P(B_R) = B_I \oplus T^b_P(B_R) \geq_k B_I.$$

Therefore, $B_R = B_I$. $\square$

To summarize, when we consider inconsistent valuations (they appear naturally, especially when we measure beliefs of groups of independent experts), we encounter an interesting phenomenon. An *inconsistent* valuation B_I, even when it is an s-model of a program, may have different justified revisions. However, all these additional revisions must be less than B_I in the

knowledge ordering. In the case of consistent models this phenomenon does not occur. If a valuation B is consistent and satisfies P then it is its own unique P-justified revision.

In the case of ordinary revision programs, "additional evidence does not destroy justified revisions" (Theorem 2.10). We will now prove a generalization of this property to the case of annotated revision programs. However, as before, we need to replace the notion of a model with that of an s-model.

Theorem 6.12 *Let P, P' be annotated revision programs. Let B_R be a P-justified revision of B_I. Let B_R be an s-model of P'. Then, B_R is a $P \cup P'$-justified revision of B_I.*

Proof.
Let $C = NC(P_{B_R}|B_I)$. Let $C' = NC((P \cup P')_{B_R}|B_I)$. Clearly, $C \leq C'$. By the definition of a justified revision $B_R = (B_I \otimes -C) \oplus C$. Hence,

$$B_R \leq B_I \oplus C \leq B_I \oplus C'.$$

By Lemma 6.2 it follows that C' satisfies the bodies of all rules in $(P \cup P')_{B_R}|B_I$. Since C' is the necessary change of $(P \cup P')_{B_R}|B_I$ we conclude that C' satisfies the heads of all rules in $(P \cup P')_{B_R}|B_I$. Reasoning as in the proof of Lemma 6.3 we find that $C' = T^b_{P \cup P'}(B_R)$.

By Theorem 6.7, B_R is an s-model of P. Therefore, by Theorem 6.4, B_R is a s-model of $P \cup P'$. Theorem 6.9 implies that B_R is a $P \cup P'$-justified revision of itself. In other words,

$$B_R = (B_R \otimes -NC((P \cup P')_{B_R}|B_R)) \oplus NC((P \cup P')_{B_R}|B_R).$$

From Lemma 6.1 it follows that $NC((P \cup P')_{B_R}|B_R) = T^b_{P \cup P'}(B_R)$. Hence,

$$B_R = (B_R \otimes -C') \oplus C'.$$

Next, let us recall that $B_R = (B_I \otimes -C) \oplus C$. Hence,

$$B_R = (((B_I \otimes -C) \oplus C) \otimes -C') \oplus C'.$$

Now, using the facts that $C \leq C'$ and $-C' \leq -C$, we get the following equalities:

$$B_R = (((B_I \otimes -C) \oplus C) \otimes -C') \oplus C' =$$

$$= ((B_I \otimes -C) \otimes -C') \oplus (C \otimes -C') \oplus C' =$$

$$= (B_I \otimes (-C \otimes -C')) \oplus C' = (B_I \otimes -C') \oplus C'$$

Thus, $B_R = (B_I \otimes -C') \oplus C'$. By the definition of justified revisions, B_R is a $P \cup P'$-justified revision of B_I. □

In case of revision programs without annotations, justified revisions satisfy the minimality principle (Theorem 2.9). Namely, P-justified revisions of a database differ from the database by as little as possible.

Before generalizing the minimality principle to the case of annotated revision programs we need to specify what we mean by the difference between $\mathcal{T}^2$-valuations.

Definition 6.5 *Let R, B be $\mathcal{T}^2$-valuations. We say that B* can be transformed into *R* via *a $\mathcal{T}^2$-valuation C if $R = (B \otimes -C) \oplus C$. We say that B* can be transformed *into R if there exists $\mathcal{T}^2$-valuation C such that B can be transformed into R via C.* △

Given two $\mathcal{T}^2$-valuations, it is not necessarily the case that one of them can be transformed into the other. Indeed, let $V_\top$ be a $\mathcal{T}^2$-valuation that assigns to each atom annotation $\top$. Let $V_\bot$ be a $\mathcal{T}^2$-valuation that assigns to each atom annotation $\bot$. Then, if the lattice consists of more than one element, $V_\top$ can not be transformed into $V_\bot$.

Definition 6.6 *Let R, B be $\mathcal{T}^2$-valuations. Let $S = \{C \mid B$ can be transformed into R via $C\}$. The* difference *diff(R, B) is*

$$\textit{diff}(R, B) = \begin{cases} \prod S, & \textit{when } S \neq \emptyset, \\ V_\top & \textit{otherwise (when } S = \emptyset). \end{cases}$$

△

The following lemma describes a useful property of a difference between $\mathcal{T}^2$-valuations. Namely, that the difference between $\mathcal{T}^2$-valuations R and B is the least (in $\leq_k$ ordering) $\mathcal{T}^2$-valuation among all C such that $R = (B \otimes -C) \oplus C$.

Lemma 6.4 *Let R, B be $\mathcal{T}^2$-valuations. Let $S = \{C \mid B$ can be transformed into R via $C\}$. If $S \neq \emptyset$, then diff$(R, B) \in S$.*

Proof.

Let $S = \{C \mid B \text{ can be transformed into } R \text{ via } C\} \neq \emptyset$. Then, $\text{diff}(R, B) = \prod S$. First, let us show that $-\prod S = \sum\{-C : C \in S\}$. On the one hand, $\prod S \leq C$ for all $C \in S$. Thus, $-\prod S \geq -C$ for all $C \in S$. Hence,

$$-\prod S \geq \sum\{-C : C \in S\}. \tag{6.4}$$

On the other hand, $\sum\{-C : C \in S\} \geq -C$ for all $C \in S$. Thus, $-\sum\{-C : C \in S\} \leq C$ for all $C \in S$. Hence, $-\sum\{-C : C \in S\} \leq \prod S$. That is,

$$\sum\{-C : C \in S\} \geq -\prod S. \tag{6.5}$$

From (6.4) and (6.5) it follows that $-\prod S = \sum\{-C : C \in S\}$.

Since $\mathcal{T}$ is complete and infinitely distributive, we get the following.

$$\begin{aligned}(B \otimes -\prod S) \oplus \prod S = (B \otimes \sum\{-C : C \in S\}) \oplus \prod S = \\ = \sum\{(B \otimes -C) : C \in S\} \oplus \prod S = \\ = \prod\{\sum\{(B \otimes -C) : C \in S\} \oplus C' : C' \in S\} \geq \\ \geq \prod\{(B \otimes -C') \oplus C' : C' \in S\} = \prod\{R\} = R.\end{aligned}$$

That is,

$$(B \otimes -\prod S) \oplus \prod S \geq R. \tag{6.6}$$

By definition of S, for each $C \in S$, $R = (B \otimes -C) \oplus C$. Therefore, for each $C \in S$, $C \leq R$ and $B \otimes -C \leq R$. Thus, $\prod S \leq R$ and

$$B \otimes -\prod S = B \otimes \sum\{-C : C \in S\} = \sum\{(B \otimes -C) : C \in S\} \leq R.$$

Hence, $(B \otimes -\prod S) \oplus \prod S \leq R$. This together with (6.6) imply that

$$(B \otimes -\prod S) \oplus \prod S = R.$$

That is, $\prod S \in S$. □

Now we will show that the minimality principle can be generalized to the case of annotated revision programs. We will have, however, to assume that $\mathcal{T}$ is a Boolean algebra and restrict ourselves to consistent $\mathcal{T}^2$-valuations.

Let $\mathcal{T}$ be a Boolean algebra with de Morgan complement being the complement. Let us define the *negation* operation on $\mathcal{T}^2$ as $\neg\langle\alpha,\beta\rangle = \langle\overline{\alpha},\overline{\beta}\rangle$ ($\alpha,\beta\in\mathcal{T}$). Then the lattice $\mathcal{T}^2$ with operations $\oplus$, $\otimes$, $\neg$, and elements $\bot$, $\top$ is a Boolean algebra, too. Operations on $\mathcal{T}^2$ lift pointwise to the space of $\mathcal{T}^2$-valuations. The space of $\mathcal{T}^2$-valuations with operations $\oplus$, $\otimes$, $\neg$, and elements $V_\bot$, $V_\top$ is again a Boolean algebra.

Lemma 6.5 *Let $\mathcal{T}$ be a Boolean algebra. Let R, B, I be $\mathcal{T}^2$-valuations. Let R and I be consistent. Let diff$(R,B)\le_k$ diff(R,I). Then, $R\otimes B\ge_k R\otimes I$.*

Proof.
Let $C=\text{diff}(R,I)$, $C'=\text{diff}(R,B)$. Since I is consistent, $I\le_k -I$. Thus,

$$I\otimes -(\neg I)\le_k -I\otimes -(\neg I) = -(I\oplus\neg I) = -V_\top = V_\bot \tag{6.7}$$

Since R is consistent, C is consistent, too. That is, $C\le_k -C$. Hence,

$$I\otimes -C = (I\otimes -C)\oplus(I\otimes C) \tag{6.8}$$

Consider valuation $C\otimes\neg I$. Using (6.7) and (6.8) we get:

$$(I\otimes -(C\otimes\neg I))\oplus(C\otimes\neg I) = (I\otimes(-C\oplus -(\neg I)))\oplus(C\otimes\neg I) =$$

$$= (I\otimes -C)\oplus(I\otimes -(\neg I))\oplus(C\otimes\neg I) = (I\otimes -C)\oplus(I\otimes C)\oplus V_\bot\oplus(C\otimes\neg I) =$$

$$= (I\otimes -C)\oplus(I\otimes C)\oplus(C\otimes\neg I) = (I\otimes -C)\oplus(C\otimes(I\oplus\neg I)) =$$

$$= (I\otimes -C)\oplus(C\otimes V_\top) = (I\otimes -C)\oplus C = R.$$

Consequently, $C\le_k C\otimes\neg I$ (by definition of diff(R,I)). Hence, $C\otimes I\le_k C\otimes\neg I\otimes I = V_\bot$. That is, $C\otimes I = V_\bot$. Since $C'\le C$, it follows that $C'\otimes I = V_\bot$. We have: $I\otimes -C\le_k R = (B\otimes -C')\oplus C'$. Thus,

$$I\otimes -C = (I\otimes -C)\otimes I\le_k((B\otimes -C')\oplus C')\otimes I = ((B\otimes -C')\otimes I)\oplus(C'\otimes I) =$$

$$= ((B\otimes -C')\otimes I)\oplus V_\bot = (B\otimes -C')\otimes I\le_k B\otimes -C'.$$

That is,

$$I\otimes -C\le_k B\otimes -C'. \tag{6.9}$$

Since R is consistent, C' is consistent, too. It means that $C' \leq_k -C'$. Hence, $B \otimes -C' \geq_k B \otimes C'$. Therefore,

$$R \otimes B = ((B \otimes -C') \oplus C') \otimes B = ((B \otimes -C') \otimes B) \oplus (C' \otimes B) =$$

$$= (B \otimes -C') \oplus (B \otimes C') = B \otimes -C'.$$

That is,

$$R \otimes B = B \otimes -C'. \tag{6.10}$$

Similarly,

$$R \otimes I = I \otimes -C. \tag{6.11}$$

Combining (6.9), (6.10), and (6.11) we get $R \otimes I \leq_k R \otimes B$. □

If $\mathcal{T}$ is *not* a Boolean algebra, then the statement of the above lemma does not necessarily hold, as illustrated by the following example.

Example 6.10 Let $\mathcal{T} = \mathcal{T}_{[0,1]}$, $U = \{a\}$. Let $R(a) = \langle 0.3, 0.7\rangle$, $B(a) = \langle 0.2, 0.5\rangle$, and $I(a) = \langle 0.1, 0.6\rangle$. Both R and I are consistent. It is easy to see that $(\text{diff}(R, B))(a) = (\text{diff}(R, I))(a) = \langle 0.3, 0.7\rangle$. Hence, $\text{diff}(R, B) \leq_k \text{diff}(R, I)$. However, $R \otimes B \not\geq_k R \otimes I$. Indeed, $(R \otimes B)(a) = \langle 0.2, 0.5\rangle$, and $(R \otimes I)(a) = \langle 0.1, 0.6\rangle$. □

Theorem 6.13 *Let R be a consistent P-justified revision of a consistent I. Let $C = diff(R, I)$. Let B be such that $diff(R, B) = C' \leq_k C$. Then, R is a P-justified revision of B.*

Proof.

Consider two reducts $P_R|I$ and $P_R|B$. Let $r' \in P_R$. Let $(l{:}\beta)$ be an annotated revision atom from the body of r'. Let $(\theta^{-1}(I))(l) = \delta_I$, $(\theta^{-1}(B))(l) = \delta_B$, and $(\theta^{-1}(R))(l) = \delta_R$. By the definition of a reduct, the corresponding rule in $P_R|I$ contains in the body the annotated revision literal $(l{:}\gamma_I)$, where $\gamma_I = \mathit{pcomp}(\delta_I, \beta)$. The corresponding rule in $P_R|B$ contains in the body the annotated revision literal $(l{:}\gamma_B)$, where $\gamma_B = \mathit{pcomp}(\delta_B, \beta)$. By the definition of pseudocomplement,

$$\delta_I \vee \gamma_I \geq \beta. \tag{6.12}$$

Since $r' \in P_R$, $\beta \leq \delta_R$. Hence, $\beta \wedge \delta_R = \beta$. Also, from the definition of *pcomp* we get $\gamma_I \leq \beta$, which implies $\gamma_I \wedge \delta_R = \gamma_I$. From (6.12) we get

$$(\delta_I \vee \gamma_I) \wedge \delta_R \geq \beta \wedge \delta_R.$$

That is,

$$(\delta_I \wedge \delta_R) \vee \gamma_I \geq \beta.$$

From Lemma 6.5 it follows that $\delta_B \wedge \delta_R \geq \delta_I \wedge \delta_R$. Therefore,

$$\delta_B \vee \gamma_I \geq (\delta_B \wedge \delta_R) \vee \gamma_I \geq \beta.$$

From definition of *pcomp*(δ_B, β) it follows that $\gamma_B \leq \gamma_I$. This means that the only difference between reducts $P_R|I$ and $P_R|B$ is that annotations of literals in the bodies of rules from $P_R|B$ are less than annotations of corresponding literals in $P_R|I$. Consequently, $NC(P_R|B) \geq_k NC(P_R|I)$.

Since R is consistent,

$$C' \leq_k C \leq_k NC(P_R|I) \leq_k NC(P_R|B) \leq_k R \leq_k$$

$$\leq_k -R \leq_k -NC(P_R|B) \leq_k -C \leq_k -C'.$$

Also, $R = (B \otimes -C') \oplus C'$ implies that $B \otimes -C' \leq_k R$, and $B \oplus C' \geq_k R$. Then, on one hand,

$$(B \otimes -NC(P_R|B)) \oplus NC(P_R|B) \leq_k (B \otimes -C') \oplus R \leq_k R \oplus R = R.$$

On the other hand,

$$(B \otimes -NC(P_R|B)) \oplus NC(P_R|B) = (B \oplus NC(P_R|B)) \otimes -NC(P_R|B) \geq_k$$

$$\geq_k (B \oplus C') \otimes R \geq_k R \otimes R = R.$$

Therefore, $(B \otimes -NC(P_R|B)) \oplus NC(P_R|B) = R$. That is, R is a P-justified revision of B. □

Theorem 6.14 *Let R be a consistent P-justified revision of a consistent I. Then, diff(R, I) is minimal in the family $\{$diff$(B, I) : B$ is a consistent model of $P\}$.*

Proof.

Let $C = \text{diff}(R, I)$. Then, $R = (I \otimes -C) \oplus C$. Since R is consistent, C is also consistent. That is, $C \leq_k -C$. Let B be a consistent model of P, and let $\text{diff}(B, I) = C' \leq_k C$. We have $B = (I \otimes -C') \oplus C'$. Inequality $C' \leq_k C$ implies $C' \leq_k C \leq_k -C \leq_k -C'$. Therefore,

$$(B \otimes -C) \oplus C = (((I \otimes -C') \oplus C') \otimes -C) \oplus C =$$

$$= (I \otimes -C' \otimes -C) \oplus (C' \otimes -C) \oplus C = (I \otimes -C) \oplus C' \oplus C =$$

$$= (I \otimes -C) \oplus C = R.$$

Consequently, $\text{diff}(R, B) \leq_k C$. By Theorem 6.13, R is a P-justified revision of B. However, B is a consistent model of P. By Theorem 6.11, B is the only P-justified revision of itself. Therefore, $R = B$. □

The condition in the above theorem that revision is consistent is important. For inconsistent revisions the minimality principle does not hold, as shown in the following example.

Example 6.11 Let $\mathcal{T} = \mathcal{T}_{\{p\}}$ with the de Morgan complement being the set-theoretic complement. Let P be an annotated revision program consisting of the following rules:

$$(\mathbf{in}(a){:}\{p\}) \leftarrow$$

$$(\mathbf{out}(a){:}\{p\}) \leftarrow (\mathbf{out}(a){:}\{p\})$$

Let $I(a) = \langle \emptyset, \{p\} \rangle$. Then I is consistent. Let $R_1(a) = \langle \{p\}, \{p\} \rangle$ and $R_2(a) = \langle \{p\}, \emptyset \rangle$. Both R_1 and R_2 are P-justified revisions of I. Thus, R_1 is an inconsistent s-model of P, and R_2 is a consistent model of P. We have: $\text{diff}(R_1, I) = \langle \{p\}, \{p\} \rangle$, and $\text{diff}(R_2, I) = \langle \{p\}, \emptyset \rangle$. Hence, $\text{diff}(R_2, I) \leq_k \text{diff}(R_1, I)$. Therefore, R_1 is a P-justified revision of a consistent I, but $\text{diff}(R_1, I)$ is *not* minimal in the family $\{\text{diff}(B, I) : B \text{ is a consistent model of } P\}$. □

6.4 An alternative way of describing annotated revision programs and the order isomorphism theorem

We will now provide an alternative description of annotated revision programs. Instead of evaluating separately *revision* atoms in $\mathcal{T}$ we will evaluate atoms in $\mathcal{T}^2$. This alternative presentation will allow us to obtain a result on the preservation of justified revisions under order isomorphisms of $\mathcal{T}^2$. This result is a generalization of the "shifting theorem" of [18].

An expression of the form $a : \langle \alpha, \beta \rangle$, where $\langle \alpha, \beta \rangle \in \mathcal{T}^2$, will be called an *annotated atom* (thus, annotated atoms are *not* annotated revision atoms). Intuitively, an atom $a : \langle \alpha, \beta \rangle$ stands for the conjunction of $(\mathbf{in}(a) : \alpha)$ and $(\mathbf{out}(a) : \beta)$. An *annotated rule* is an expression of the form $p \leftarrow q_1, \ldots, q_n$ where $p, q_1, \ldots, q_n$ are annotated atoms. An *annotated program* is a set of annotated rules.

A $\mathcal{T}^2$-valuation B *satisfies* an annotated atom $a:\langle\alpha,\beta\rangle$ if $\langle\alpha,\beta\rangle \leq_k B(a)$. This notion of satisfaction can be extended to annotated rules and annotated programs.

We will now define the notions of reduct, necessary change and justified revision for the new kind of programs. Let P be an annotated program. Let B_I and B_R be two $\mathcal{T}^2$-valuations. The reduct of a program P with respect to two valuations B_I and B_R is defined in a manner similar to Definition 6.2. Specifically, we leave only the rules with bodies that are satisfied by B_R, and in the remaining rules we reduce the annotated atoms (except that now the transformation θ is no longer needed!).

Definition 6.7 *The* reduct $P_{B_R}|B_I$ *is obtained from* P *by*

1. *removing every rule whose body contains an annotated atom that is not satisfied in* B_R*,*
2. *replacing each annotated atom* $l{:}\beta$ *from the body of each remaining rule by the annotated atom* $l{:}\gamma$*, where* $\gamma = pcomp(B_I(l),\beta)$ *(here* $\beta,\gamma \in \mathcal{T}^2$*).* $\triangle$

Next, we compute the least fixpoint of the operator associated with the reduced program. Finally, as in Definition 6.3, we define the concept of justified revision of a valuation B_I with respect to a revision program P.

Definition 6.8 B_R *is a* P-justified revision *of* B_I *if* $B_R = (B_I \otimes -C) \oplus C$*, where* $C = NC(P_{B_R}|B_I)$ *is the necessary change for* $P_{B_R}|B_I$*.* $\triangle$

It turns out that this new syntax does not lead to a new notion of justified revision. Since we talk about two different syntaxes, we will use the term "old syntax" to denote the revision programs as defined in Section 6.1, and "new syntax" to describe programs introduced in this section. Specifically we now exhibit two mappings. The first of them, tr_1, assigns to each "old" in-rule

$$(\mathbf{in}(a){:}\alpha) \leftarrow (\mathbf{in}(b_1){:}\alpha_1), \ldots, (\mathbf{in}(b_m){:}\alpha_m), (\mathbf{out}(s_1){:}\beta_1), \ldots, (\mathbf{out}(s_n){:}\beta_n),$$

a "new" rule

$$a{:}\langle\alpha,\bot\rangle \leftarrow b_1{:}\langle\alpha_1,\bot\rangle, \ldots, b_m{:}\langle\alpha_m,\bot\rangle, s_1{:}\langle\bot,\beta_1\rangle, \ldots, s_n{:}\langle\bot,\beta_n\rangle.$$

An "old" out-rule

$$(\mathbf{out}(a){:}\beta) \leftarrow (\mathbf{in}(b_1){:}\alpha_1), \ldots, (\mathbf{in}(b_m){:}\alpha_m), (\mathbf{out}(s_1){:}\beta_1), \ldots, (\mathbf{out}(s_n){:}\beta_n)$$

is encoded in analogous way:

$$a{:}\langle \bot, \beta\rangle \leftarrow b_1{:}\langle \alpha_1, \bot\rangle, \ldots, b_m{:}\langle \alpha_m, \bot\rangle, s_1{:}\langle \bot, \beta_1\rangle, \ldots, s_n{:}\langle \bot, \beta_n\rangle.$$

Translation tr_2, in the other direction, replaces a "new" revision rule by one in-rule and one out-rule. Specifically, a "new" rule

$$a{:}\langle \alpha, \beta\rangle \leftarrow a_1{:}\langle \alpha_1, \beta_1\rangle, \ldots, a_n{:}\langle \alpha_n, \beta_n\rangle$$

is replaced by two "old" rules (with identical bodies but different heads)

$$(\mathbf{in}(a){:}\alpha) \leftarrow (\mathbf{in}(a_1){:}\alpha_1), (\mathbf{out}(a){:}\beta_1), \ldots, (\mathbf{in}(a_n){:}\alpha_n), (\mathbf{out}(a_n){:}\beta_n)$$

and

$$(\mathbf{out}(a){:}\beta) \leftarrow (\mathbf{in}(a_1){:}\alpha_1), (\mathbf{out}(a){:}\beta_1), \ldots, (\mathbf{in}(a_n){:}\alpha_n), (\mathbf{out}(a_n){:}\beta_n).$$

The translations tr_1 and tr_2 can be extended to programs. We then have the following theorem that states that the new syntax and semantics of annotated revision programs presented in this section are equivalent to the syntax and semantics introduced and studied in Section 6.3.

Theorem 6.15 *Both transformations tr_1, and tr_2 preserve justified revisions. That is, if B_I, B_R are valuations in $\mathcal{T}^2$ and P is a program in the "old" syntax, then B_R is a P-justified revision of B_I if and only if B_R is a $tr_1(P)$-justified revision of B_I. Similarly, if B_I, B_R are valuations in $\mathcal{T}^2$ and P is a program in the "new" syntax, then B_R is a P-justified revision of B_I if and only if B_R is a $tr_2(P)$-justified revision of B_I.*

In the case of unannotated revision programs, the shifting theorem proved in [18] shows that for every revision program P and every two initial databases B and B' there is a revision program P' such that there is a one-to-one correspondence between P-justified revisions of B and P'-justified revisions of B'. In particular, it follows that the study of justified revisions (for unannotated programs) can be reduced to the study of justified revisions of empty databases. We will now present a counterpart of this result for annotated revision programs. The situation here is more complex. It

is no longer true that a $\mathcal{T}^2$-valuation can be "shifted" to any other $\mathcal{T}^2$-valuation. However, the shift is possible if the two valuations are related to each other by an order isomorphism of the lattice of all $\mathcal{T}^2$-valuations.

There are many examples of order isomorphisms on the lattice of $\mathcal{T}^2$. For instance, the mapping $\psi : \mathcal{T}^2 \to \mathcal{T}^2$ defined by $\psi(\langle\alpha, \beta\rangle) = \langle\beta, \alpha\rangle$ is an order isomorphism of $\mathcal{T}^2$. In the case of the lattice $\mathcal{T}_X$, order isomorphisms of $\mathcal{T}_X^2$ can also be generated by permutations of the set X.

Let ψ be an order isomorphism on $\mathcal{T}^2$. It can be extended to annotated atoms, annotated rules, and $\mathcal{T}^2$-valuations as follows:

$\psi(a : \delta) = a : \psi(\delta)$,

$\psi(a{:}\delta \leftarrow a_1{:}\delta_1, \ldots, a_n{:}\delta_n) = \psi(a{:}\delta) \leftarrow \psi(a_1{:}\delta_1), \ldots, \psi(a_n{:}\delta_n)$,

$(\psi(B))(a) = \psi(B(a))$,

where $a, a_1, \ldots, a_n \in U$, $\delta, \delta_1, \ldots, \delta_n \in \mathcal{T}^2$, and B is a $\mathcal{T}^2$-valuation.

The extension of an order isomorphism on $\mathcal{T}^2$ to $\mathcal{T}^2$-valuations is again an order isomorphism, this time on the lattice of all $\mathcal{T}^2$-valuations. We say that an order isomorphism ψ on a lattice *preserves conflation* if $\psi(-\delta) = -\psi(\delta)$ for all elements δ from the lattice. We now have the following result that generalizes the Shifting Theorem (Theorem 3.4).

Theorem 6.16 *Let ψ be an order isomorphism on the set of $\mathcal{T}^2$-valuations. Let ψ preserve conflation. Then, B_R is a P-justified revision of B_I if and only if $\psi(B_R)$ is a $\psi(P)$-justified revision of $\psi(B_I)$.*

Proof.

By definition, B_R is a P-justified revision of B_I if and only if $B_R = (B_I \otimes -C) \oplus C$, where $C = NC(P_{B_R}|B_I)$. Since ψ is an order isomorphism, it preserves meet and join operations. Therefore,

$$\psi(B_R) = \psi((B_I \otimes -C) \oplus C) = \psi(B_I \otimes -C) \oplus \psi(C) =$$

$$= (\psi(B_I) \otimes \psi(-C)) \oplus \psi(C) = (\psi(B_I) \otimes -\psi(C)) \oplus \psi(C).$$

At the same time, $\psi(P_{B_R}|B_I) = (\psi(P))_{\psi(B_R)}|\psi(B_I)$, and $NC(\psi(P_{B_R}|B_I)) = \psi(NC(P_{B_R}|B_I))$. Thus, B_R is a P-justified revision of B_I if and only if $\psi(B_R)$ is a $\psi(P)$-justified revision of $\psi(B_I)$.

□

The Shifting Theorem (Theorem 3.4) which applies to ordinary revision programs is just a particular case of Theorem 6.16. In order to derive it from Theorem 6.16, we take $\mathcal{T} = \mathcal{TWO}$. Next, we consider an ordinary revision program P and two databases B_1 and B_2 (let us recall that in the case of ordinary revision programs, databases are *sets of atoms* and not valuations). Let P^a and B_1^v and B_2^v be defined as in Theorem 6.6. It is easy to see that the operator ψ, defined by

$$(\psi(v))(a) = \begin{cases} \langle \beta, \alpha \rangle, & \text{when } B_1^v(a) \neq B_2^v(a) \\ \langle \alpha, \beta \rangle, & \text{when } B_1^v(a) = B_2^v(a) \end{cases},$$

is an order-isomorphism on $\mathcal{TWO}^2$-valuations and that $\psi(B_1^v) = B_2^v$. Let C_1 and C_2 be two sets of atoms such that $C_2^v = \psi(C_1^v)$. By Theorem 6.16, C_1^v is a P^a-justified revision of B_1^v if and only if C_2^v is a $\psi(P^a)$-justified revision of B_2^v. Theorem 6.6 and the observation that the necessary change of $P^a_{C_1^v} | B_1^v$ is consistent if and only if the necessary change of $\psi(P^a)_{C_2^v} | B_2^v$ is consistent together imply now the Shifting Theorem (Theorem 3.4).

The requirement in Theorem 6.16 that ψ preserves conflation is essential. If it is not the case, the statement of the theorem may not hold, as illustrated by the following example.

Example 6.12 Let $\mathcal{T} = \mathcal{T}_{\{p,q,r\}}$ with the de Morgan complement defined as follows:

$$\overline{\{\}} = \{p,q,r\}, \quad \overline{\{p\}} = \{p,r\}, \quad \overline{\{q\}} = \{q,r\}, \quad \overline{\{r\}} = \{p,q\},$$
$$\overline{\{p,q,r\}} = \{\}, \quad \overline{\{p,r\}} = \{p\}, \quad \overline{\{q,r\}} = \{q\}, \quad \overline{\{p,q\}} = \{r\}.$$

Let ψ be an order isomorphism on $\mathcal{T}$ such that $\psi(\{p\}) = \{p\}$, $\psi(\{q\}) = \{r\}$, and $\psi(\{r\}) = \{q\}$. Then, ψ does not preserve conflation, because

$$\psi(-\langle\{p\},\{\}\rangle) = \psi(\langle\{p,q,r\},\{p,r\}\rangle) = \langle\{p,q,r\},\{p,q\}\rangle, \text{ but}$$

$$-\psi(\langle\{p\},\{\}\rangle) = -\langle\{p\},\{\}\rangle = \langle\{p,q,r\},\{p,r\}\rangle.$$

Let an annotated program be the following:

$$P: \qquad a : \langle\{p\},\{\}\rangle \;\leftarrow$$

It determines the necessary change $C(a) = \langle\{p\},\{\}\rangle$.

Then, $-C(a) = \langle\{p,q,r\},\{p,r\}\rangle$. Let $B_I(a) = \langle\{\},\{r\}\rangle$. The P-justified revision of B_I is $B_R(a) = (\langle\{\},\{r\}\rangle \otimes \langle\{p,q,r\},\{p,r\}\rangle) \oplus \langle\{p\},\{\}\rangle = \langle\{p\},\{r\}\rangle$.

The annotated program $\psi(P)$ is the same as P. We have $\psi(B_I)(a) = \langle\{\}, \{q\}\rangle$, $\psi(B_R)(a) = \langle\{p\}, \{q\}\rangle$. The reduct $(\psi(P))_{\psi(B_R)}|\psi(B_I) = \psi(P) = P$. The necessary change determined by the reduct is C. However,

$$((\psi(B_I) \otimes -C) \oplus C)(a) = \langle\{p\}, \{\}\rangle \neq \psi(B_R)(a).$$

Therefore, $\psi(B_R)$ is *not* a $\psi(P)$-justified revision of $\psi(B_I)$. □

6.5 Disjunctive annotated revision programs.

A *disjunctive annotated revision rule* is expression of the form

$$p_1|\ldots|p_m \leftarrow q_1, \ldots, q_n, \tag{6.13}$$

where $p_1, \ldots, p_m, q_1, \ldots, q_n$ are annotated revision atoms.

A *disjunctive annotated revision program* is a collection of disjunctive revision rules.

A $\mathcal{T}$-valuation v is *closed* under a disjunctive annotated revision rule 6.13 if v satisfies at least one of $p_1, \ldots, p_m$, or v does not satisfy at least one of $q_1, \ldots, q_n$.

A $\mathcal{T}$-valuation v is *closed* under a disjunctive annotated revision program $\mathcal{P}$ if it is closed under all rules of $\mathcal{P}$.

Let $\mathcal{P}$ be a disjunctive annotated revision program. A *necessary change* entailed by $\mathcal{P}$ is any minimal in the $\leq_k$ ordering $\mathcal{T}$-valuation that is closed under $\mathcal{P}$.

Definition 6.9 *Let $\mathcal{P}$ be a disjunctive annotated revision program and let B_I and B_R be $(\mathcal{T} \odot \mathcal{T})$-valuations. The* reduct *of $\mathcal{P}$ with respect to (B_I, B_R) (denoted $\mathcal{P}^n_{B_R}|B_I$) is defined as follows.*

1. *Remove from $\mathcal{P}$ every rule whose body contains an annotated revision atom that is not satisfied in B_R.*

2. *Replace each annotated atom $l : \beta$ from the body of each remaining rule by the annotated atom $l : \gamma$, where $\gamma = \gamma(\theta^{-1}(B_I)(l), \beta)$.* △

Definition 6.10 *B_R is a* $\mathcal{P}$-justified revision *of B_I if $B_R = (B_I \otimes -C) \oplus C$, where C is some necessary change for $\mathcal{P}^n_{B_R}|B_I$.* △

Theorem 6.17 *Let $\mathcal{P}$ be an annotated revision program (without disjunctions in the heads of the rules). Let B_R, B_I be $(\mathcal{T} \odot \mathcal{T})$-valuations. Then, B_R is a $\mathcal{P}$-justified revision of B_I if and only if B_R is a $\mathcal{P}$-justified revision of B_I when $\mathcal{P}$ is treated as a disjunctive annotated revision program.*

Chapter 7

Conclusions and Future Work

The thesis is devoted to revision programming, a knowledge representation formalism to describe and enforce constraints on databases. Problems studied in the thesis include connections of revision programs with logic programs, well-founded semantics for revision programs, extensions of revision programming formalism, annotated revision programs, computing justified revisions.

The thesis establishes a foundation for further research in applications and implementations in the area of revision programming and implies a number of open problems.

We developed three ways of defining well-founded semantics for revision programming (Chapter 4). We showed that the procedure described in Section 4.2.1 in some cases gives better results than WFS^{PT} and WFS^{Sh}. Whether it always outperforms WFS^{PT} and WFS^{Sh} is yet to be established. Well-founded semantics can be computed in polynomial time. Therefore, it can serve as a preprocessing tool for computing justified revisions. This makes a problem of finding a better well-founded semantics important since it can lead to better implementations.

The two algorithms for implementation of justified revisions described in Section 3.1.3 and Section 3.2.4 of the thesis involve embedding revision programs into logic programs and then using tools for computing stable models. Another possible direction in which this work may continue is to write an implementation specific to revision programming that does not involve translation to logic programs. Enhanced with a good well-founded semantics this implementation may outperform existing implementations for solving problems that can be represented in the formalism of revision programming.

A natural area of applications for revision programming is that of database repairs. However, not much research has been done in this direction and that remains to be studied.

Revision programs describe constraints known at a particular time that need to be imposed on a database. After the change is applied there may be another set of constraints to satisfy, and so on. This give rise to a question whether revisions can be iterated. The problem of iterated updates is studied by other authors (for example, LUPS work by Alferes, Pereira, Przymusinski in [3]). The question whether revision programming can be used to represent iterative updates needs investigation.

References

[1] J.J. Alferes, J.A. Leite, L.M. Pereira, H. Przymusinska, and T.C. Przymusinski. Dynamic logic programming. In *Principles of Knowledge Representation and Reasoning. Proceedings of the 6th International Conference, KR'98, Trento, Italy, June 2-5, 1998*, pages 98–111. Morgan Kaufmann, 1998.

[2] 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.

[3] J.J. Alferes, L.M. Pereira, H. Przymusinska, and T.C. Przymusinski. LUPS – a language for updating logic programs. In *Logic Programming and Nonmonotonic Reasoning, 5th International Conference, LPNMR'99*, volume 1730 of *Lecture Notes in Computer Science*, pages 162–176. Springer-Verlag, 1999.

[4] K.R. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of deductive databases and logic programming. Papers from the workshop held in Washington, D.C., August 18–22, 1986*, pages 89–148, Palo Alto, CA, 1988. Morgan Kaufmann.

[5] T. J. M. Bench-Capon. *Knowledge Representaion. An Approach to Artificial Intelligence*. Academic Press Inc., 1990.

[6] K. Berman, J. Schlipf, and J.Franco. Computing the well-founded semantics faster. In *Logic Programming and Nonmonotonic Reasoning (Lexington, KY, 1995)*, volume 928 of *Lecture Notes in Computer Science*, pages 113–125, Berlin, 1995. Springer.

[7] 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.

[8] M. C. Fitting. Fixpoint semantics for logic programming – a survey. To appear in Theoretical Computer Science.

[9] M. C. Fitting. Annotated revision specification programs. In *Logic programming and nonmonotonic reasoning (Lexington, KY, 1995)*, volume 928 of *Lecture Notes in Computer Science*, pages 143–155, Berlin, 1995. Springer.

[10] 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.

[11] M.L. Ginsberg. Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[12] H.A. Kautz and B. Selman. Hard problems for simple default logics. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (Toronto, ON, 1989)*, Morgan Kaufmann Series in Representation and Reasoning, pages 189–197, San Mateo, CA, 1989. Morgan Kaufmann.

[13] M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12(4):335–367, 1992.

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

[15] V. Lifschitz, L. R. Tang, and H. Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999.

[16] 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.

[17] W. Marek, I. Pivkina, and M. Truszczyński. Annotated revision programs. In *Logic Programming and Nonmonotonic Reasoning, 5th International Conference, LPNMR'99*, volume 1730 of *Lecture Notes in Computer Science*, pages 49–62. Springer-Verlag, 1999.

[18] W. Marek, I. Pivkina, and M. Truszczyński. Revision programming = logic programming + integrity constraints. In *Computer Science Logic, 12th International Workshop, CSL'98*, volume 1584 of *Lecture Notes in Computer Science*, pages 73–89. Springer-Verlag, 1999.

[19] W. Marek, I. Pivkina, and M. Truszczyński. Annotated revision programs. Accepted for publication in Artificial Intelligence Journal, 2000.

[20] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38(3):588–619, 1991.

[21] 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.

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

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

[24] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[25] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In M.J. Maher, editor, *Proceedings of the 1996 Joint International Conference and Syposium on Logic Programming (JICSLP-96) (Bonn, Germany, September 2-6, 1996)*, pages 289–303. MIT Press, 1996.

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

[27] H. Rasiowa and R. Sikorski. *The Mathematics of metamathematics*. PWN—Polish Scientific Publishers, Warsaw, 1970.

[28] 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.

[29] T. Syrjanen. Implementation of local grounding for logic programs with stable model semantics. Technical Report B18, Digital Systems Laboratory, Helsinki University of Technology, October 1998.

[30] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[31] A. Tarski. *Logic, semantics, metamathematics*. Oxford at the Clarendon Press, Oxford, 1956.

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

[33] 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.

[34] A. Van Gelder, K.A. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1988) (March 21-23, 1988, Austin, Texas)*, pages 221–230, New York, 1988. ACM Press.

[35] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.