

UNIVERSITÀ DELLA CALABRIA



Dipartimento di ELETTRONICA,  
INFORMATICA E SISTEMISTICA

**UNIVERSITÀ DELLA CALABRIA**

**Dipartimento di Elettronica,  
Informatica e Sistemistica**

**Dottorato di Ricerca in  
Ingegneria dei Sistemi e Informatica  
XX ciclo**

***Tesi di Dottorato***

**Declarative Semantics for  
Consistency Maintenance**

**Luciano Caroprese**



UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,  
Informatica e Sistemistica

Dottorato di Ricerca in  
Ingegneria dei Sistemi e Informatica  
XX ciclo

*Tesi di Dottorato*

Declarative Semantics for  
Consistency Maintenance

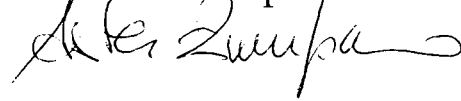
Luciano Caroprese



Coordinatore  
Prof. Domenico Talia



Supervisore  
Prof. Ester Zumpano



DEIS

To my family...

“If the facts don’t fit the theory, change the facts.”

*Albert Einstein*

---

## **Acknowledgements**

I would like to thank my advisor Prof. Ester Zumpano for her guidance and continuous support during these three years.

I would also like to thank Prof. Sergio Greco for his valuable advice and suggestions.

I would like to express my gratitude to Prof. Mirosław Truszczyński who has been so deeply involved in our research and who, at the same time, has helped me to explore those aspects of the work which I found particularly interesting.

I wish to thank my parents for their patience and affection.

Finally, special thanks to all my friends for the various forms of help and encouragement that they have given me.



---

## Contents

<b>1</b>	<b>Preliminaries</b> .....	3
1.1	Introduction .....	3
1.2	Logic Programs .....	4
1.3	Queries .....	6
1.4	Complexity Classes .....	7
1.5	Complexity of Datalog Queries .....	10
<b>2</b>	<b>Techniques for Repairing and Querying</b> .....	11
2.1	Introduction .....	11
2.2	Computing Repairs .....	13
2.3	Querying Database using Logic Programs with Exceptions .....	16
2.4	Query Answering in the Presence of Constraints .....	18
2.5	Complete Answers from Incomplete Databases .....	22
2.6	Condensed Representation of Database Repairs for Consistent Query Answering .....	24
2.7	Using Views .....	30
2.8	Minimal Change Integrity Maintenance using Tuple Deletions .....	38
<b>3</b>	<b>Active Integrity Constraints</b> .....	41
3.1	Introduction .....	41
3.1.1	Contribution .....	43
3.1.2	Plan of the Chapter .....	43
3.2	Databases and Integrity constraints .....	44
3.2.1	Integrity Constraints .....	44
3.2.2	Repairing and Querying Inconsistent Databases .....	44
3.2.3	Repairing and Querying through Stable Models .....	46

3.3	Active Integrity Constraints .....	47
3.4	Computation and Complexity .....	54
3.4.1	Rewriting into Logic Programs .....	55
3.4.2	Data Complexity .....	57
3.4.3	Preferred Repairs and Answers .....	59
<b>4</b>	<b>Active Integrity Constraints and Revision Programming</b> .....	<b>61</b>
4.1	Introduction .....	61
4.1.1	Contribution .....	62
4.1.2	Plan of the Chapter .....	63
4.2	Weak Repairs and Founded Weak Repairs .....	63
4.3	Justified Repairs .....	65
4.4	Normal Active Integrity Constraints and Normalization .....	70
4.5	Shifting Theorem .....	72
4.6	Complexity and Computation .....	77
4.7	Some Implications of the Results Obtained so far .....	81
4.8	Connections between Revision Programs and Active Integrity Constraints .....	82
4.8.1	Revision Programming — an Overview .....	82
4.8.2	Proper Revision Programs .....	86
4.8.3	Revision Programs as Sets of Active Integrity Constraints ..	87
4.8.4	Shifting Theorem for Revision Programs .....	89
4.9	Computation and Complexity Results for Revision Programming ..	91
<b>5</b>	<b>View Updating through Active Integrity Constraints</b> .....	<b>93</b>
5.1	Introduction .....	93
5.1.1	Contribution .....	95
5.1.2	Plan of the Chapter .....	96
5.2	A Declarative Semantics for View Updating .....	96
5.3	Rewriting into Active Integrity Constraints .....	100
5.4	Soundness, Completeness and Complexity Results .....	105
5.5	Related Works .....	106
<b>6</b>	<b>Conclusions</b> .....	<b>109</b>
	<b>References</b> .....	<b>111</b>



---

## Introduction

Integrity constraints are conditions on databases. If a database violates integrity constraints, it needs to be *repaired* so that the integrity constraints hold again. Often there are several ways to enforce integrity constraints. To illustrate the problem of database repair with respect to integrity constraints, let us consider the database  $\mathcal{I} = \{a, b\}$  and the integrity constraint  $\text{not } a \vee \text{not } b$  stating that the database does not contain  $a$  or it does not contain  $b$ . Clearly,  $\mathcal{I}$  does not satisfy the integrity constraint and needs to be *repaired* – replaced by a database that satisfies the constraint. Assuming that the possible facts that the database could contain are  $\{a, b, c, d\}$ , the databases  $\emptyset$ ,  $\{a\}$ ,  $\{b\}$ ,  $\{a, c\}$  are examples of databases that could be considered as replacements for  $\mathcal{I}$ . Since the class of replacements of  $\mathcal{I}$  is quite large, the question arises whether there is a principled way to narrow it down. One of the most intuitive and commonly accepted postulates is that the change between the initial database and the revised database be minimal. In our case, the minimality of change narrows down the class of possible revisions to  $\{a\}$  and  $\{b\}$ . The minimality of change leads to the concept of *repair*, a minimal set of update actions (*insertions* and *deletions* of facts) that makes the database consistent.

In some cases, the minimality of change is not specific enough and may leave too many candidate solutions. The problem can be addressed by formalisms that allow the database designer to formulate integrity constraints and, in addition, to state preferred ways for enforcing them. In this thesis, we study two such formalisms: *active integrity constraints* and *revision programming*.

Essentially, an active integrity constraint is an integrity constraint that specifies the update actions that can be performed when it is violated. It is composed by a conjunction of literals, the *body*, that represents a *condition* that should be *false* and by a disjunction of update actions, the *head*, that can be performed when the body is *true* (that is when the constraint is violated). The active integrity constraints work in a domino-like manner as the satisfaction of one of them may trigger the violation and therefore the activation of another one. The first semantics for active integrity constraints here introduced, allows us to identify, among all possible repairs, those whose actions are *specified* in the head of some active integrity constraint and *sup-*

ported by the database or by other updates. These repairs are called *founded repairs*. We show that the computation of founded repairs can be done by rewriting the constraints into a Datalog program and then computing its stable models; each stable model will represent a founded repair.

Next, we compare *active integrity constraints* and *revision programming*, another formalisms designed to describe integrity constraints on databases and to specify *preferred* ways to enforce them. The original semantics proposed for these formalisms differ. The semantics for revision programs defines the concept of *justified revision*. The thesis shows that despite the differences in the syntax, and the lack of a simple correspondence between justified revisions and founded repairs, the two frameworks are closely related.

A justified revision is a set of *revision literals*, an alternative way to model updates over a database, that can be inferred by means of the revision program and by the set of all atoms that do not change their state of *presence (in)* or *absence (out)* during the update process. We show that each founded repair corresponds to a justified revision, but not vice-versa. Next, we broaden the class of semantics for the two formalisms by introducing a different semantics for active integrity constraints and a different semantics for revision programs. The first one allows us to compute a smaller set of repairs, the *justified repairs*, that correspond to justified revisions. The second one allow us to compute a wider set of revision, the *founded revisions*, that correspond to founded repairs. The introduction of these new semantics aligns the two formalisms showing that each of them is a notational variants of the other. We show that for each semantics the *shifting property* holds. Shifting consists of transforming an instance of a database repair problem to another syntactically isomorphic instance by changing active integrity constraints or revision programs to reflect the “shift” from the original database to the new one.

Finally, the thesis defines a formal declarative semantics for view updating in the presence of existentially derived predicates and non-flat integrity constraints, that translates an update request against a view into an update of the underlying database. The new semantics allows to identify, among the set of all possible repairs, the subset of *supported repairs*, that is repairs whose actions are validated by the database or by other updates. Given a deductive database and an update request, the computation of supported repairs is performed by rewriting the update request and the deductive database in the form of active integrity constraints.

## Preliminaries

**Summary.** In this chapter we introduce some basic concepts on logic programming and computational complexity. For a detailed treatment see [10, 11, 86, 105, 106, 117, 125, 142]. We briefly introduce syntax of (disjunctive) logic programs and present the stable model semantics. Moreover, after introducing complexity measures, we survey complexity results of various forms of logic programming.

### 1.1 Introduction

The theory of deductive databases begins with Codd's paper [41] in which the formal definition of the *relational model* was given. A relational database consists of a set of *facts* whose contents can be used to answer queries. The need for deducting new information from the facts already present in the database and the necessity to deal with incomplete information leads to the concept of *deductive databases*. A deductive database, in addition to storing individual facts (extensional data), stores deductive rules (intensional data) that are used to answer queries.

*Logic programming* was introduced by Kowalski [86] and the first Prolog interpreter was implemented by Roussel in 1972 [42]. Logic programming introduced the concept of *declarative* contrast to *procedural* programming. Ideally, based on Kowalski's *principle of separation of logic and control* [88], a programmer should only be concerned with the *declarative meaning* of the program, while the procedural aspects of the execution should be handled automatically. The formal definition of logic programming starts with the classical paper by Van Emden and Kowalski on the least model semantics [142], Reiter's paper on the closed world assumption [126] and Lloyd's "*Foundations of Logic Programming*" [105].

The connection between logic programming and deductive databases [106, 125] quickly became clear and leads to a *logical approach to knowledge representation*. This approach is based on the idea of providing intelligent machines with a *logical specification* of the knowledge they possess; hence a precise *meaning* or *semantics*

has to be associated with any logic or database program in order to give its declarative specification.

Finding a suitable semantics of deductive databases and logic programs is one of the most important and difficult research problems.

## 1.2 Logic Programs

### Syntax

By an alphabet  $\Sigma$  of a first order language we mean a (finite or countably infinite) set of *variables*, *predicates* and *constants*. In the following, we use the letters  $p, q, r, \dots$  for predicate symbols,  $X, Y, Z, \dots$  for variables and  $a, b, c, \dots$  for constants. A *term* a constant or a variable. An *atom* over  $\Sigma$  is a formula  $p(t_1, \dots, t_n)$ , where  $p$  is a predicate symbol of arity  $n$  and  $t_i$ 's are terms.

The *first order language*  $\mathcal{L}$  over the alphabet  $\Sigma$  is defined as the set of all well-formed first order formulae that can be built starting from the atoms and using connectives, quantifiers and punctuation symbols in a standard way.

A *literal*  $L$  is an atom  $A$  or a negated atom *not*  $A$ ; in the former case, it is *positive*, and in the latter *negative*. Two literals are *dual*, if they are of the form  $A$  and *not*  $A$ , for some atom  $A$ . Given a literal  $L$ , we write  $L^D$  for its dual. The dual operator is extended to sets of literals as appropriate.

A (*disjunctive Datalog*) *rule*  $r$  is a clause of the form <sup>1</sup>

$$\bigvee_{i=1}^p A_i \leftarrow \bigwedge_{j=1}^m B_j, \bigwedge_{j=m+1}^n \text{not } B_j, \varphi \quad p + n > 0 \quad (1.1)$$

where  $A_1, \dots, A_p, B_1, \dots, B_n$  are *atoms* and  $\varphi$  is a conjunction of built-in atoms of the form  $u \theta v$  such that  $u$  and  $v$  are terms and  $\theta$  is a comparison predicate. The set  $\{A_1, \dots, A_p\}$  is the *head* of  $r$  (denoted by  $\text{head}(r)$ ), while the set  $\{B_1, \dots, B_m, \text{not } B_{m+1}, \dots, \text{not } B_n, \varphi\}$  is the *body* of  $r$  (denoted by  $\text{body}(r)$ ). It is assumed that each rule is *safe* [139], i.e. that a variable appearing in the head or in a negative literal also appears in a positive body literal.

A rule with a ground atom in the head and an empty body and is called *fact*. In this case the symbol ' $\leftarrow$ ' can be omitted. The expression  $\mathcal{H} \leftarrow \mathcal{B}_1 \vee \dots \vee \mathcal{B}_n$  can be used as shorthand for the rules  $\mathcal{H} \leftarrow \mathcal{B}_1, \dots, \mathcal{H} \leftarrow \mathcal{B}_n$ .

If a rule is *not*-free (resp.  $\vee$ -free) it is called *positive* (resp. *normal*). A logic program  $\mathcal{P}$  consists of a finite set of rules. It is *positive* (resp. *normal*) if all its rules are positive (resp. normal).

<sup>1</sup> A literal can appear in a conjunction or in a disjunction at most once. The meaning of the symbols ' $\wedge$ ' and ' $\vee$ ' is the same.

Given a program  $\mathcal{P}$ , some of the predicate symbols are defined by a number of facts and do not occur in the head of any other rule. They are called *base* or *EDB predicates*. The other predicate symbols are called *derived* or *IDB predicates*.

The *Herbrand Universe*  $U_{\mathcal{P}}$  of a program  $\mathcal{P}$  is the set of all constants appearing in  $\mathcal{P}$ , and its *Herbrand Base*  $\mathcal{B}_{\mathcal{P}}$  is the set of all ground atoms constructed from the predicates appearing in  $\mathcal{P}$  and the constants from  $U_{\mathcal{P}}$ . A term (resp. an atom, a literal, a rule or a program) is *ground* if no variables occur in it. A rule  $r'$  is a *ground instance* of a rule  $r$ , if  $r'$  is obtained from  $r$  by replacing every variable in  $r$  with some constant in  $U_{\mathcal{P}}$ ;  $ground(\mathcal{P})$  denotes the set of all ground instances of the rules in  $\mathcal{P}$ . We assume that  $ground(\mathcal{P})$  does not contain built-in atoms in the body of rules, as *true* built-in atoms can be deleted and rules with *false* built-in atoms are deleted from  $ground(\mathcal{P})$ .

### Stable Model Semantics

An interpretation of  $\mathcal{P}$  is any subset of  $\mathcal{B}_{\mathcal{P}}$ . The value of a ground atom  $A$  with respect to an interpretation  $\mathcal{I}$ ,  $value_{\mathcal{I}}(A)$ , is *true* if  $A \in \mathcal{I}$  and *false* otherwise. The value of a ground negated literal *not*  $A$  is *true* if  $A \notin \mathcal{I}$  and *false* otherwise. Given a set  $S$  of ground literals,

$$value_{\mathcal{I}}^{and}(S) = \begin{cases} \min(\{value_{\mathcal{I}}(L) \mid L \in S\}) & \text{if } S \neq \emptyset \\ true & \text{if } S = \emptyset \end{cases}$$

and

$$value_{\mathcal{I}}^{or}(S) = \begin{cases} \max(\{value_{\mathcal{I}}(L) \mid L \in S\}) & \text{if } S \neq \emptyset \\ false & \text{if } S = \emptyset \end{cases}$$

A ground rule  $r$  is *satisfied* by  $\mathcal{I}$  if  $value_{\mathcal{I}}^{or}(head(r)) \geq value_{\mathcal{I}}^{and}(body(r))$ . Thus, a rule  $r$  with empty body is satisfied by  $\mathcal{I}$  if  $value_{\mathcal{I}}^{or}(head(r)) = true$ . In the following the existence of rules with an empty head which define *denials* is also assumed, that is rules which are satisfied only if the body is *false* ( $value_{\mathcal{I}}^{and}(body(r)) = false$ ). An interpretation  $\mathcal{M}$  for  $\mathcal{P}$  is a model of  $\mathcal{P}$  if  $\mathcal{M}$  satisfies all rules in  $ground(\mathcal{P})$ . The (model-theoretic) semantics for positive  $\mathcal{P}$  assigns to  $\mathcal{P}$  the set of its *minimal models*  $\mathcal{MM}(\mathcal{P})$ , where a model  $\mathcal{M}$  for  $\mathcal{P}$  is minimal, if no proper subset of  $\mathcal{M}$  is a model for  $\mathcal{P}$ . The more general *disjunctive stable model semantics* also applies to programs with (unstratified) negation [66]. Disjunctive stable model semantics generalizes stable model semantics, previously defined for normal programs [65]. For any interpretation  $\mathcal{M}$ , denote with  $\mathcal{P}^{\mathcal{M}}$  the ground positive program derived from  $ground(\mathcal{P})$  by 1) removing all rules that contain a negative literal *not*  $A$  in the body and  $A \in \mathcal{M}$ , and 2) removing all negative literals from the remaining rules. An interpretation  $\mathcal{M}$  is a stable model of  $\mathcal{P}$  if and only if  $\mathcal{M} \in \mathcal{MM}(\mathcal{P}^{\mathcal{M}})$ . For general  $\mathcal{P}$ , the stable model semantics assigns to  $\mathcal{P}$  the set  $\mathcal{SM}(\mathcal{P})$  of its *stable models*. It is well known that stable models are minimal models (that is  $\mathcal{SM}(\mathcal{P}) \subseteq \mathcal{MM}(\mathcal{P})$ ) and that for negation free programs, minimal and stable model semantics coincide (that is  $\mathcal{SM}(\mathcal{P}) = \mathcal{MM}(\mathcal{P})$ ). Observe that stable models are minimal models which are “supported”, that is their atoms can be derived from the program. For instance, the

program consisting of the rule  $a \vee b \leftarrow \text{not } c$  has three minimal models  $\mathcal{M}_1 = \{a\}$ ,  $\mathcal{M}_2 = \{b\}$  and  $\mathcal{M}_3 = \{c\}$ . However, only  $\mathcal{M}_1$  and  $\mathcal{M}_2$  are stable.

### Stratified Programs

A normal logic program  $\mathcal{P}$  is said to be *stratified* [9, 64, 99] if it is possible to decompose the set  $S$  of its predicate symbols into disjoint sets  $S_1, S_2, \dots, S_n$ , called *strata*, so that for every clause

$$r : C \leftarrow A_1, \dots, A_m, \text{not } B_1, \dots, \text{not } B_n \quad (n \geq 0)$$

in  $\mathcal{P}$ , where  $A$ 's,  $B$ 's and  $C$  are atoms, we have that:

1. for every  $i$ ,  $\text{stratum}(A_i) \leq \text{stratum}(C)$  and
2. for every  $j$ ,  $\text{stratum}(B_j) < \text{stratum}(C)$ ,

where  $\text{stratum}(A) = \alpha$ , if the predicate symbol of  $A$  belongs to  $S_\alpha$ . Any particular decomposition  $S_1, S_2, \dots, S_n$  satisfying the above conditions is called a *stratification* of  $\mathcal{P}$ . A program  $\mathcal{P}$  is called *stratified* if it has a stratification.

Stratification assigns relative priorities between ground atoms so that priority conflicts (cycles) can be avoided and meaningless semantics are discarded. Stratified programs allow a disciplined form of negation, that is when using negation we can refer to an already defined relation, so that the definition is not circular, or as Van Gelder puts it, negation through recursion is avoided.

Stratifiability is easy to check by constructing the *dependency graph*. Given a program  $\mathcal{P}$  the dependency graph  $\mathcal{D}_{\mathcal{P}}$  consists of the predicate names as the vertices  $\langle p_i, p_j, s \rangle$ ; there is a labeled edge in  $\mathcal{D}_{\mathcal{P}}$  iff there is a rule  $r$  in  $\mathcal{D}$  with  $p_i$  in its head and  $p_j$  in its body and the label  $s \in \{+, -\}$  denoting whether  $p_j$  appears in a positive or a negative literal body of  $r$ . A cycle in the dependency graph is said to be a negative cycle if it contains at least one edge with a negative label. A normal logic program  $\mathcal{P}$  is stratified if its dependency graph  $\mathcal{D}_{\mathcal{P}}$  does not contain any negative cycle. Stratified programs *have a unique stable model*, called *perfect model*. Obviously, for positive logic programs the perfect model is equivalent to the least model.

Moreover, a program is called *semipositive* if negation is only applied to base atoms. Clearly, semipositive programs are also stratified.

## 1.3 Queries

Given a database  $\mathcal{I}$ , a program  $\mathcal{P}$  and a predicate symbol  $g$ ,  $\mathcal{P}_{\mathcal{I}}$  denotes the program derived from the union of  $\mathcal{P}$  with the facts in  $\mathcal{I}$ , that is  $\mathcal{P}_{\mathcal{I}} = \mathcal{P} \cup \mathcal{I}$  and  $\mathcal{I}(g)$  denotes the set of  $g$ -facts in  $\mathcal{I}$  that is the facts in  $\mathcal{I}$  whose predicate symbol is  $g$ . The semantics of  $\mathcal{P}_{\mathcal{I}}$  is given by the set of its stable models by considering either their union (*possible semantics* or *brave reasoning*) or their intersection (*certain semantics*).

or *cautious reasoning*). A disjunctive Datalog query  $Q$  is a pair  $(g, \mathcal{P})$  where  $g$  is a predicate symbol, called the *query goal*, and  $\mathcal{P}$  is a disjunctive Datalog program. The answer to a disjunctive Datalog query  $Q = (g, \mathcal{P})$  over a database  $\mathcal{I}$ , denoted as  $Q(\mathcal{I})$ , under the possible (resp. certain) semantics, is given by  $\mathcal{I}'(g)$  where  $\mathcal{I}' = \bigcup_{\mathcal{M} \in \mathcal{SM}(\mathcal{P}_{\mathcal{I}})} \mathcal{M}$  (resp.  $\mathcal{I}' = \bigcap_{\mathcal{M} \in \mathcal{SM}(\mathcal{P}_{\mathcal{I}})} \mathcal{M}$ ).

## 1.4 Complexity Classes

We assume that the reader is familiar with the basic notions of complexity classes [80, 117]. In this section we give a brief survey of the standard complexity classes, following the notation given in [80]. A complete account can be found in [117].

*Turing machines.*

A *Turing machine* is a device consisting of a semi-infinite *tape* which can be read and write. Formally, a *deterministic Turing machine* ( $\mathcal{DTM}$ )  $\mathbf{T}$  is a quadruple  $(S, \Sigma, \delta, s_0)$ , where  $S$  is a finite set of states,  $\Sigma$  is a finite alphabet of *symbols*,  $\delta$  is the *transition function* and  $s_0 \in S$  is the *initial state*. The alphabet contains a special symbol called *blank* and represented as  $\sqcup$ . The transition function  $\delta$  is a map:

$$\delta : S \times \Sigma \rightarrow (S \cup \{\text{halt}, \text{yes}, \text{no}\}) \times \Sigma \times \{-1, 0, 1\}$$

where *halt*, *yes* and *no* denote three additional states not occurring in  $S$ , whereas  $\{-1, 0, 1\}$  denote the *motion directions*. The tape is divided into *cells* containing symbols of  $\Sigma$ , and a *cursor* may move along the tape. The input string  $\mathcal{I}$  is written on the input tape. The machine takes successive *steps* of computation according to  $\delta$ , and when any of the states *halt*, *yes* or *no* is reached. We say that  $\mathbf{T}$  accepts the input  $\mathcal{I}$  if it halts in ‘yes’, rejects it if it halts in ‘no’, while if the *halt* state is reached we say that the output of  $\mathbf{T}$  is computed.

A *non deterministic Turing machine*  $\mathcal{NDTM}$  is a quadruple  $(S, \Sigma, \Delta, s_0)$ , where  $S, \Sigma, s_0$  are the same as before, while the possible operations the machine can perform are described no longer by a function, but by the relation:

$$\Delta \subseteq S \times \Sigma \times (S \cup \{\text{halt}, \text{yes}, \text{no}\}) \times \Sigma \times \{-1, 0, 1\}$$

In contrast to a  $\mathcal{DTM}$ , now the definition of acceptance and rejection is asymmetric. We say that a  $\mathcal{NDTM}$  *accepts* an input if there is at least one sequence of choices leading to the state ‘yes’, and it *rejects* an input if no sequence of choices can lead to ‘yes’.

*Time and space bounds.*

The *time* expended by a  $\mathcal{DTM}$   $\mathbf{T}$  on an input  $\mathcal{I}$  is defined as the number of steps taken by  $\mathbf{T}$  on  $\mathcal{I}$  from the start to the halting. Note that if  $\mathbf{T}$  does not halt on  $\mathcal{I}$ , then

the time is considered to be infinite. The *time* expended by a  $\mathcal{NDTM}$  on an input  $\mathcal{I}$  is 1 if  $\mathbf{T}$  does not accept the  $\mathcal{I}$ , otherwise it is defined as the minimum over the number of steps in any accepting computation of  $\mathbf{T}$ . The *space* required by a  $\mathcal{DTM}$   $\mathbf{T}$  on an input  $\mathcal{I}$  is defined as the number of cells visited by the cursor during the computation. For a  $\mathcal{NDTM}$  the space is defined as 1 if  $\mathbf{T}$  does not accept the  $\mathcal{I}$ , otherwise it is defined as the minimum over the number of cells on the tape over all accepting computations of  $\mathbf{T}$ .

Let  $\mathbf{T}$  be a  $\mathcal{DTM}$  or a  $\mathcal{NDTM}$  and  $f$  a function from the positive integers to themselves, we say that:

- $\mathbf{T}$  halts *in time*  $O(f(n))$  if there exist positive integers  $c$  and  $n_0$  such that the time the time expended by  $\mathbf{T}$  on any input of length  $n$  is not greater than  $cf(n)$  for all  $n \geq n_0$ .
- $\mathbf{T}$  halts *within space*  $O(f(n))$  if the space required by  $\mathbf{T}$  on any input of length  $n$  is not greater than  $cf(n)$  for all  $n \geq n_0$ , where  $c$  and  $n_0$  are positive integers.

Let  $\Sigma$  be a finite alphabet containing  $\sqcup$ , let  $\Sigma' = \Sigma \setminus \{\sqcup\}$ , and  $\mathcal{L} \subseteq \Sigma'^*$  a language in  $\sigma'$ , that is a set of finite strings over  $\Sigma'$ . Let  $\mathbf{T}$  be a  $\mathcal{DTM}$  or a  $\mathcal{NDTM}$  such that (i) if  $x \in \mathcal{L}$  then  $\mathbf{T}$  accepts  $x$ ; (ii) if  $x \notin \mathcal{L}$  then  $\mathbf{T}$  rejects  $x$ . Then we say that  $T$  *decides*  $\mathcal{L}$ . Moreover (i) if  $T$  halts in time  $O(f(n))$  we say  $\mathbf{T}$  *decides*  $\mathcal{L}$  *in time*  $O(f(n))$ , (ii) if  $\mathbf{T}$  halts within space  $O(f(n))$  we say  $\mathbf{T}$  *decides*  $\mathcal{L}$  *within space*  $O(f(n))$ .

Given a function  $f$  on positive integers the set of languages are defined as follows:

- $\text{TIME}(f(n)) = \{\mathcal{L} \mid \mathcal{L} \text{ is decided by some } \mathcal{DTM} \text{ in time } O(f(n))\}$
- $\text{NTIME}(f(n)) = \{\mathcal{L} \mid \mathcal{L} \text{ is decided by some } \mathcal{NDTM} \text{ in time } O(f(n))\}$
- $\text{SPACE}(f(n)) = \{\mathcal{L} \mid \mathcal{L} \text{ is decided by some } \mathcal{DTM} \text{ within space } O(f(n))\}$
- $\text{NSPACE}(f(n)) = \{\mathcal{L} \mid \mathcal{L} \text{ is decided by some } \mathcal{NDTM} \text{ within space } O(f(n))\}$

All the previous sets are *complexity classes*. The complexity classes of most interest are not classes corresponding to particular functions, but their union. The following abbreviations denote the main complexity classes:

- $\text{P} = \bigcup_{d>0} \text{TIME}(n^d)$
- $\text{NP} = \bigcup_{d>0} \text{NTIME}(n^d)$
- $\text{EXPTIME} = \bigcup_{d>0} \text{TIME}(2^{n^d})$
- $\text{NEXPTIME} = \bigcup_{d>0} \text{NTIME}(2^{n^d})$
- $\text{PSPACE} = \bigcup_{d>0} \text{SPACE}(n^d)$
- $\text{EXPSPACE} = \bigcup_{d>0} \text{SPACE}(2^{n^d})$
- $\text{L} = \bigcup_{d>0} \text{SPACE}(\log n)$
- $\text{NL} = \bigcup_{d>0} \text{NSPACE}(\log n)$

Note that the class  $\text{EXPTIME}$  and  $\text{NEXPTIME}$  can be viewed as 1- $\text{EXPTIME}$  and 1- $\text{NEXPTIME}$  respectively, where 1 means the first level of the exponentiation. Double exponents are captured by the classes 2- $\text{EXPTIME}$ , 3- $\text{EXPTIME}$  and so on, defined as:

$$\bigcup_{d>0} \text{TIME}(2^{2^{n^d}}), \bigcup_{d>0} \text{TIME}(2^{2^{2^{n^d}}}), \dots$$

The relations among the complexity classes are as follows:



1.  $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$
2.  $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$
3.  $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$
4.  $\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log(n)+f(n)})$
5.  $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$
6.  $\text{TIME}(f(n)) \subset \text{TIME}((f(2n+1))^3)$  (*Theorem of the temporal hierarchy*)
7.  $\text{SPACE}(f(n)) \subset \text{SPACE}(f(n) \cdot \log(n))$  (*Theorem of the spatial hierarchy*)

From the above properties the following hierarchy holds:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$$

Note that at least one of the four inclusions is strict for the theorem of the spatial hierarchy.

Any class  $C$  has its *complementary class* denoted as  $\text{co-}C$  and defined as follows: for any language  $\mathcal{L}$  in  $\Sigma'$ , let  $\bar{\mathcal{L}}$  denote its *complement*, that is  $\text{co-}C$  is  $\{\bar{\mathcal{L}} \mid \mathcal{L} \in C\}$ .

Let  $\mathcal{L}_1$  and  $\mathcal{L}_2$  be languages. Assume that there is a  $\mathcal{DTM} \mathbf{T}$  such that

- For all input strings  $\mathcal{I}$ , we have  $\mathcal{I} \in \mathcal{L}_1$  if and only if  $\mathbf{T}(\mathcal{I}) \in \mathcal{L}_2$ , where  $\mathbf{T}(\mathcal{I})$  denotes the output of  $\mathbf{T}$  on input  $\mathcal{I}$ .
- $\mathbf{T}$  halts within space  $O(\log n)$ .

Then  $\mathbf{T}$  is called a *logarithmic-space reduction* from  $\mathcal{L}_1$  to  $\mathcal{L}_2$  and we say that  $\mathcal{L}_1$  is *reducible* to  $\mathcal{L}_2$ . Let  $C$  be a set of languages. A language  $\mathcal{L}$  is called *C-hard* if any language  $\mathcal{L}'$  in  $C$  is reducible to  $\mathcal{L}$ . If  $\mathcal{L}$  is *C-hard* and  $\mathcal{L} \in C$  then  $\mathcal{L}$  is called *C-complete*.

*The polynomial hierarchy.*

In order to define the polynomial hierarchy we introduce the oracle Turing machines. Given a language  $A$  an *oracle*  $\mathcal{DTM} \mathbf{T}^A$ , also called a *DTM with oracle*  $A$ , is an ordinary  $\mathcal{DTM}$  augmented by an additional *query tape* and additional three states: *query*,  $\in$ ,  $\notin$ . When  $\mathbf{T}^A$  is in the state *query* the computation proceeds as usual,  $\mathbf{T}^A$  changes its state from *query* to  $\in$  or  $\notin$  depending whether the string present on the query tape belongs to  $A$  or not; when  $\mathbf{T}^A$  reaches the state  $\in$  or  $\notin$  the query tape is instantaneously erased.

Thus let  $C$  be a set of languages, the complexity classes  $P^C$  and  $NP^C$  are defined as follows. For a language  $\mathcal{L}$ , it is  $\mathcal{L} \in P^C$  (or  $\mathcal{L} \in NP^C$ ) iff there is some language  $A \in C$  and some polynomial-time oracle  $\mathcal{DTM}$  (or  $\mathcal{NDTM}$ )  $\mathbf{T}^A$  such that  $\mathbf{T}^A$  decides  $\mathcal{L}$ .

The polynomial hierarchy consists of the classes  $\Delta_k^P, \Sigma_k^P, \Pi_k^P$  (con  $k \geq 0$ ) defined as follows:

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P;$$

For  $k > 0$ :

$$\Delta_{k+1}^P = P^{\Sigma_k^P} ; \Sigma_{k+1}^P = NP^{\Sigma_k^P} ; \Pi_{k+1}^P = \text{co-}\Sigma_{k+1}^P.$$

Thus,  $\Delta_{k+1}^P$  is the set of languages decided in polynomial time by some  $\mathcal{DTM}$  with an oracle for  $\Sigma_k^P$  problems. Analogously  $\Sigma_{k+1}^P$  is the set of languages decided

in polynomial time by some  $\mathcal{NDTM}$  with an oracle for  $\Sigma_k^P$  problems. Finally, the class  $\Pi_k^P$  contains all the languages whose complement is in  $\Sigma_k^P$ . Note that  $\Delta_1^P = P^P = P$ ,  $\Sigma_1^P = NP^P = NP$  and  $\Pi_1^P = co-\Sigma_1^P = co-NP$ . Moreover  $\Delta_2^P = P^{NP}$ , that is  $\Delta_2^P$  is the set of languages that can be decided by calling polynomial times an oracle NP.

Previous classes define a polynomial hierarchy in which the following properties hold:

1.  $\Delta_k^P \supseteq (\Sigma_{k-1}^P \cup \Pi_{k-1}^P)$ ;
2.  $\Delta_k^P \subseteq (\Sigma_k^P \cap \Pi_k^P)$ .

The set of classes in the polynomial hierarchy is denoted as  $\mathcal{PH}$  (*polynomial hierarchy*) and is defined as  $\mathcal{PH} = \bigcup_{k \geq 0} \Sigma_k^P$ .

## 1.5 Complexity of Datalog Queries

We refer to a query  $Q = (A, \mathcal{P})$  over a database  $\mathcal{I}$  under possible (resp. certain) semantics, asking whether an atom  $A$  belongs to some stable model (resp. all stable models) of  $\mathcal{P} \cup \mathcal{I}$ . There are two main kinds of complexity connected to this problem [144]:

- The **data complexity** is the complexity of the problem when  $\mathcal{P}$  is *fixed* while  $\mathcal{I}$  and  $A$  are the *input*.
- The **program complexity** is the complexity of the problem when  $\mathcal{I}$  and  $A$  are *fixed* while  $\mathcal{P}$  is the *input*.

The complexity of this problem depends on  $\mathcal{P}$  [44]. If we consider data complexity, the problem is:

- P-complete for both semantics if  $\mathcal{P}$  is *normal and positive*;
- P-complete for both semantics if  $\mathcal{P}$  is *normal and stratified*;
- NP-complete (resp. coNP-complete) under possible semantics (resp. certain semantics) if  $\mathcal{P}$  is *normal*;
- $\Sigma_2^P$ -complete (resp.  $\Pi_2^P$ -complete) under possible semantics (resp. certain semantics) if  $\mathcal{P}$  is *disjunctive*;

While if we consider program complexity, the problem is:

- EXPTIME-complete for both semantics if  $\mathcal{P}$  is *normal and positive*;
- EXPTIME-complete for both semantics if  $\mathcal{P}$  is *normal and stratified*;
- NEXPTIME-complete (resp. co-NEXPTIME-complete) under possible semantics (resp. certain semantics) if  $\mathcal{P}$  is *normal*.
- NEXPTIME<sup>NP</sup>-complete (resp. co-NEXPTIME<sup>NP</sup>-complete) under possible semantics (resp. certain semantics) if  $\mathcal{P}$  is *disjunctive*.

## Techniques for Repairing and Querying

**Summary.** This chapter gives an informal description of the main techniques for repairing and querying inconsistent databases proposed in the literature. After a brief introduction to these techniques, some of them will be described in more details.

### 2.1 Introduction

Logic programming based approaches enabling the computation of repairs and consistent answers in a possibly inconsistent database have been proposed in [12, 13, 39, 40, 70, 71, 145].

In [13] a logical characterization of the notion of consistent answer in a possibly inconsistent database is introduced. The technique is based on the computation of an equivalent query, obtained by using the notion of residue developed in the context of semantic query optimization, derived from the original source query. In [12] an approach consisting in the use of a Logic Program with Exceptions (LPe) for obtaining consistent query answers is proposed. The semantics of a LPe is obtained from the semantics for Extended Logic Programs, by adding extra conditions that assign higher priority to exceptions. The method, given a set of integrity constraints and an inconsistent database instance, consists in the direct specification of database repairs in a logic programming formalism. Both the techniques in [13] and [12] have been shown to be complete for universal binary integrity constraints and universal quantified queries.

In [70, 71] a general framework for computing repairs and consistent answers over inconsistent databases with universally quantified variables has been proposed. The technique is based on the rewriting of constraints into extended disjunctive rules with two different forms of negation (negation as failure and classical negation). The disjunctive program can be used for two different purposes: to compute ‘repairs’ for the database, and to produce consistent answers. The technique has been shown to be sound and complete and more general than previously proposed techniques. A detailed description of this approach will be provided in Chapter 3.

In [145] a general framework for repairing databases consisting in correcting faulty values within the tuples, without actually deleting them (value-based approach), is proposed. Repairs are represented by using *trustable tableau* that allows conjunctive queries to be answered efficiently.

In [40] a practical framework for computing consistent query answers for large, possibly inconsistent relational databases is proposed. The proposed framework handles union of conjunctive queries and can effectively (and efficiently) extract indefinite disjunctive information from an inconsistent database. In [39] the problem of minimal-change integrity maintenance in the context of integrity constraints (denial constraints, general functional and inclusion dependencies, as well as key and foreign key constraints) in relational databases, has been investigated.

The recent literature on consistent query answering is reviewed in [37]. Specifically, the paper discusses some computational and semantic limitations of consistent query answering, and summarizes selected research directions in this area.

Other works have investigated the updating of data and knowledge bases through the use of nonmonotonic formalisms [7, 8, 109, 111]. In [7] the problem of updating knowledge bases represented by logic programs has been investigated. More specifically, the authors introduce the notion of updating a logic program by means of another logic program and a new paradigm, called *dynamic logic programming*, to model dynamic program update. The new paradigm has been further investigated in [8], where the language LUPS (Language for Dynamic Updates), designed for specifying changes to logic programs, has been proposed. Given an initial knowledge base (in the form of a logic program) LUPS provides a way for sequentially updating it. The declarative meaning of a sequence of sets of update actions in LUPS is defined using the semantics of the dynamic logic program generated by those actions.

In [109, 111] revision programming, a logic-based framework for describing constraints on databases and providing a computational mechanism to enforce them, is introduced. Revision programming, based on the extension of the logic programming paradigm, captures those constraints that can be stated in terms of the membership (presence or absence) of atoms in a database. Such a constraint is represented by a revision rule  $\alpha \leftarrow \alpha_1, \dots, \alpha_k$  where  $\alpha$  and all  $\alpha_i$  are of the form **in**( $a$ ) and **out**( $b$ ). A *revision program* is a collection of revision rules whose semantics, called *justified revision semantics*, assigns to any database  $\mathcal{I}$  a (possibly empty) collection of *justified revisions* of  $\mathcal{I}$ . A justified revision models a set of *revision literals* updates over the inconsistent database that can be inferred by means of the revision program and by the set of all atoms that do not change their state of presence or absence during the update process.

A detailed description of this approach will be provided in Chapter 4.

In [110] the work of Fitting [60] that assigns annotations to revision atoms so that providing a way to quantify the confidence (probability) that a revision atom holds is re-examined. In particular, starting from the observation that this semantics does not always provide results consistent with intuition, an alternative treatment of annotated

revision programs is proposed by changing both the notion of a model of a program and the notion of a justified revision.

Postulates for update and revision operators for knowledge bases have been discussed in [81], whereas the problem of belief revision has been addressed in [79, 98]. In such a framework update consists in bringing the knowledge base up to date when the world described by it changes, whereas revision is used when new information is obtained about a static world. The computational complexity of several update operators, proposed in the literature, has been discussed in [50] (see also [19] for the special problem of updating the knowledge of agents). All these works do not take into account the possibility of indicating the update operations to make the database consistent.

Another category of approaches proposed in the literature, for the automatic maintenance of databases, uses ECA (Event-Condition-Action) rules for checking and enforcing integrity constraints. The application of the ECA paradigm of active databases to policies—collection of general principles specifying the desired behavior of systems—has been investigated in [38]. A framework for enforcing constraints by issuing actions to be performed to correct violations has been proposed in [32] and [33]. Policies for database maintenance using situation calculus have been studied in [22], whereas the problem of maintaining integrity constraints in database systems has been considered in [114] where an algorithm for automatically transforming an integrity constraint into a set of active rules has been proposed.

In the rest of this chapter we will provide an informal description of the main techniques for repairing and querying inconsistent databases.

## 2.2 Computing Repairs

An interesting technique has been proposed in [13]. The technique introduces a logical characterization of the notion of consistent answer in a possibly inconsistent database. Queries are assumed to be given in prefix disjunctive normal form.

A query  $Q(X)$  is a prenex disjunctive first order formula of the form:

$$K[\bigvee_{i=1}^s (\bigwedge_{j=1}^{m_i} \mathcal{P}_{i,j}(U_{i,j}) \wedge \bigwedge_{j=1}^{n_i} \neg R_{i,j}(V_{i,j}) \wedge \Psi_i)]$$

where  $K$  is a sequence of quantifiers,  $\Psi_i$  contains only built-in predicates and  $X$  denotes the list of variables in the formula.

Given a query  $Q(X)$  and a set of integrity constraints  $\eta$  a tuple  $t$  is a *consistent answer* to the query  $Q(X)$  over a database instance  $\mathcal{I}$ , written  $(Q, \mathcal{I}) \models_c t$ , if  $t$  is a substitution for the variables in  $X$  such that for each repair  $\mathcal{I}'$  of  $\mathcal{I}$ ,  $(Q, \mathcal{I}') \models_c t$ .

*Example 2.1.* Consider the relation *Student* with schema  $(Code, Name, Faculty)$  with the attribute *Code* as key. The functional dependencies:

$$\begin{aligned} Code &\rightarrow Name \\ Code &\rightarrow Address \end{aligned}$$

can be expressed by the following two constraints:

$$\begin{aligned} \forall(X, Y, Z, U, V) [Student(X, Y, Z) \wedge Student(X, U, V) \supset Y = U] \\ \forall(X, Y, Z, U, V) [Student(X, Y, Z) \wedge Student(X, U, V) \supset Z = V] \end{aligned}$$

Assume to have the following inconsistent instance of the relation *Student*:

<i>Code</i>	<i>Name</i>	<i>Faculty</i>
<i>s</i> <sub>1</sub>	<i>Mary</i>	<i>Engeneering</i>
<i>s</i> <sub>2</sub>	<i>John</i>	<i>Science</i>
<i>s</i> <sub>2</sub>	<i>Frank</i>	<i>Engeneering</i>

Student

The previous inconsistent database admits two different repairs, say *Repair*<sub>1</sub> and *Repair*<sub>2</sub>, which are reported in the following.

<i>Code</i>	<i>Name</i>	<i>Faculty</i>
<i>s</i> <sub>1</sub>	<i>Mary</i>	<i>Engeneering</i>
<i>s</i> <sub>2</sub>	<i>John</i>	<i>Science</i>

*Repair*<sub>1</sub>

<i>Code</i>	<i>Name</i>	<i>Faculty</i>
<i>s</i> <sub>1</sub>	<i>Mary</i>	<i>Engeneering</i>
<i>s</i> <sub>2</sub>	<i>Frank</i>	<i>Engeneering</i>

*Repair*<sub>2</sub>

The consistent answers to the query  $\exists Z(Student(s_1, Y, Z))$  is “Engineering”, while there is no consistent answer to the query  $\exists Z(Student(s_2, Y, Z))$ .  $\square$

*General approach.*

The technique is based on the computation of an equivalent query  $T_\omega(Q)$  derived from the source query  $Q$ . The definition of  $T_\omega(Q)$  is based on the notion of residue developed in the context of semantic query optimization.

More specifically, for each literal  $B$ , appearing in some integrity constraint, a residue  $Res(B)$  is computed. Intuitively,  $Res(B)$  is a universal quantified first order formula which must be true, because of the constraints, if  $B$  is true. Universal constraints can be rewritten as denials.

Let  $A$  be a literal,  $r$  a denial of the form  $\leftarrow B_1 \wedge \dots \wedge B_n$ ,  $B_i$  (for some  $1 \leq i \leq n$ ) a literal unifying with  $A$  and  $\theta$  the most general unifier for  $A$  and  $B_i$  such that variables in  $A$  are used to substitute variables in  $B_i$ , but they are not substituted by other variables. Then, the residue of  $A$  with respect to  $r$  and  $B_i$  is:

$$\begin{aligned} Res(A, r, Bi) &= not(B_1 \wedge \dots \wedge B_{i-1} \wedge B_{i+1} \wedge \dots \wedge B_n) \theta \\ &= not B_1 \theta \vee \dots \vee not B_{i-1} \theta \vee not B_{i+1} \theta \vee \dots \vee not B_n \theta \end{aligned}$$

The residue of  $A$  with respect to  $r$  is  $Res(A, r) = \bigwedge_{B_i | A=B_i \theta} Res(A, r, B_i)$  consisting of the conjunction of all the possible residues of  $A$  in  $r$ , whereas the residue of  $A$  with respect to a set of integrity constraints  $\eta$  is  $Res(A) = \bigwedge_{r \in \eta} Res(A, r)$ .

Thus, the residue of a literal  $A$  is a first order formula which must be true if  $A$  is true.

The operator  $T_\omega(Q)$  is defined as follows:

- $T_0(Q) = Q$ ;
- $T_i(Q) = T_{i-1}(Q) \wedge R$  where  $R$  is a residue of some literal in  $T_{i-1}$ .

The operator  $T_\omega$  represents the fixpoint of  $T$ .

It has been shown that the operator  $T$  has a fixpoint for universal quantified queries and universal binary integrity constraints, that is constraints, which written in disjunctive format, are of the form:  $\forall X(B_1 \vee B_2 \vee \theta)$  where  $B_1, B_2$  are literals and  $\theta$  is a conjunctive formula with built-in operators.

*Example 2.2.* Consider the database  $\mathcal{I}$ :

$$\mathcal{I} = \{Supply(c_1, d_1, i_1), Supply(c_2, d_2, i_2), Class(i_1, t), Class(i_2, t)\}$$

with the integrity constraint, defined by the following first order formula:

$$\forall(X, Y, Z)[Supply(X, Y, Z) \wedge Class(Z, t) \supset X = c_1]$$

stating that only supplier  $c_1$  can supply items of type  $t$ . The database  $\mathcal{I}$  is inconsistent because the integrity constraint is not satisfied (an item of type  $t$  is also supplied by supplier  $c_2$ ). The constraint can be rewritten as:

$$\leftarrow Supply(X, Y, Z) \wedge Class(Z, t) \wedge X \neq c_1$$

where all variables are (implicitly) universally quantified. The residue of the literals appearing in the constraint are:

$$\begin{aligned} Res(Supply(X, Y, Z)) &= not Class(Z, t) \vee X = c_1 \\ Res(Class(Z, t)) &= not Supply(X, Y, Z) \vee X = c_1 \end{aligned}$$

The iteration of the operator  $T$  to the query goal  $Class(Z, t)$  gives:

- $T_0(Class(Z, t)) = Class(Z, t)$ ,
- $T_1(Class(Z, t)) = Class(Z, t) \wedge (not Supply(X, Y, Z) \vee X = c_1)$ ,
- $T_2(Class(Z, t)) = Class(Z, t) \wedge (not Supply(X, Y, Z) \vee X = c_1)$ ,

At Step 2 a fixpoint is reached since the literal  $Class(Z, t)$  has been ‘expanded’ and the literal  $not Supply(X, Y, Z)$  does not have a residue associated to it. Thus, to answer the query  $Q = Class(Z, t)$  with the above integrity constraint, the query  $T_\omega(Q) = Class(Z, t) \wedge (not Supply(X, Y, Z) \vee X = c_1)$  is evaluated. The computation of  $T_\omega(Q)$  over the above database gives the result  $Z = i_1$ .  $\square$

The following example shows a case in which the technique proposed is not complete.

*Example 2.3.* Consider the integrity constraint

$$(X, Y, Z)[p(X, Y) \wedge p(X, Z) \supset Y = Z]$$

the database  $\mathcal{I} = \{p(a, b), p(a, c)\}$  and the query  $Q = \exists p(a, U)$  (we are using the formalism used in [13]). The technique proposed generates the new query

$$T_\omega(Q) = \exists U[p(a, U) \wedge Z(\neg p(a, Z) \vee Z = U)]$$

which is not satisfied contradicting the expected answer which is *true*.

This technique has also been shown to be complete for universal binary integrity constraints and universal quantified queries. Moreover the detection of fixpoint conditions is, generally, not easy.  $\square$

### 2.3 Querying Database using Logic Programs with Exceptions

The new approach proposed by Arenas-Bertossi-Chomicki in [12] consists in the use of a Logic Program with Exceptions (LPe) for obtaining consistent query answers. An LPe is a program with the syntax of an extended logic program (ELP), that is, in it we may find both logical (or strong) negation ( $\neg$ ) and procedural negation (not). In this program, rules with a positive literal in the head represent a sort of general default, whereas rules with a logically negated head represent exceptions. The semantic of an LPe is obtained from the semantics for ELP's, by adding extra conditions that assign higher priority to exceptions. The method, given a set of integrity constraints  $\eta$  and an inconsistent database instance, consists in the direct specification of database repairs in a logic programming formalism. The resulting program will have both negative and positive exceptions, strong and procedural negations, and disjunctions of literals in the head of some of the clauses; that is it will be a disjunctive extended logic program with exceptions. As in [13] the method considers a set of integrity constraints  $\eta$  written in the standard format

$$\bigvee_{i=1}^n P_i(X_i) \vee \bigvee_{i=1}^m \neg Q_i(Y_i) \vee \varphi$$

where  $\varphi$  is a formula containing only built-in predicates, and there is an implicit universal quantification in front. This method specifies the repairs of the database  $\mathcal{I}$  that violate  $\eta$ , by means of a logical program with exceptions  $\Pi^{\mathcal{I}}$ . In  $\Pi^{\mathcal{I}}$  for each predicate  $P$  a new predicate  $P'$  is introduced and each occurrence of  $P$  is replaced by  $P'$ . More specifically,  $\Pi^{\mathcal{I}}$  is obtained by introducing:

1. **Persistence Defaults.** For each base predicate  $P$ , the method introduces the persistence defaults:



$$\begin{aligned} P'(x) &\leftarrow P(x) \\ \neg P'(x) &\leftarrow \text{not } P(x) \end{aligned}$$

The predicate  $P'$  is the repaired version of the predicate  $P$ , so it contains the tuples corresponding to  $P$  in a repair of the original database.

2. **Stabilizing Exceptions.** From each integrity constraint  $ic$  and for each negative literal  $\text{not } Q_{i_0}$  in  $ic$ , the following negative exception clause is introduced:

$$\neg Q'_{i_0}(y_{i_0}) \leftarrow \bigwedge_{i=1..n} \neg P'_i(x_i), \bigwedge_{i \neq i_0} Q'_i(y_i), \varphi'$$

where  $\varphi'$  is a formula that is logically equivalent to the logical negation of  $\varphi$ . Similarly, for each positive literal  $P_{i_1}$  in the constraint it is generated the following positive exception clause:

$$P'_{i_1}(x_{i_1}) \leftarrow \bigwedge_{i \neq i_1} \neg P'_i(x_i), \bigwedge_{i=1..m} Q'_i(y_i), \varphi$$

The meaning of the Stabilizing Exceptions is to make the integrity constraints be satisfied by the new predicates. These exceptions are necessary, but not sufficient to ensure that the changes the original subject should be subject to, in order to restore consistency, are propagated to the new predicates.

3. **Triggering Exceptions.** From the integrity constraint in standard form it is produced the following disjunctive exception clause:

$$\bigvee_{i=1..n} P'_i(x_i) \vee \bigvee_{i=1..m} \neg Q'_i(y_i) \leftarrow \bigwedge_{i=1..n} \text{not } P_i(x_i), \bigwedge_{i=1..m} Q_i(y_i), \varphi'$$

The program  $\Pi^{\mathcal{I}}$  constructed as shown above is a ‘disjunctive extended repair logic program with exceptions for the database instance  $\mathcal{I}$ ’. In  $\Pi^{\mathcal{I}}$  positive defaults are blocked by negative conclusions, and negative defaults, by positive conclusions.

*Example 2.4.* Consider the database  $\mathcal{I} = \{p(a), q(b)\}$  with the inclusion dependency  $\eta$ :

$$p(X) \supset q(X)$$

In order to specify the database repairs the new predicates  $p'$  and  $q'$  are introduced. The resulting repair program has four default rules expressing that  $p'$  and  $q'$  contain exactly what  $p$  and  $q$  contain, resp.:

$$\begin{aligned} p'(x) &\leftarrow p(x) \\ q'(x) &\leftarrow q(x) \\ \neg p'(x) &\leftarrow \text{not } p(x) \quad \text{and} \\ \neg q'(x) &\leftarrow \text{not } q(x) \end{aligned}$$

two stabilizing exceptions :

$$\begin{aligned} q'(x) &\leftarrow p'(x) \\ \neg p'(x) &\leftarrow \neg q'(x) \end{aligned}$$

and the triggering exception:

$$\neg p'(x) \vee q'(x) \leftarrow p(x), \text{not } q(x)$$

The e-answer sets are:

$$\begin{aligned} \{p(a), q(b), p'(a), q'(b), \neg p'(a)\} \\ \{p(a), q(b), p'(a), q'(b), q'(b)\} \end{aligned}$$

that correspond to the two expected database repairs. □

The method can be applied to a set of domain independent binary integrity constraints, that is a set that can be checked with respect to satisfaction by looking to the active domain, and such that in each constraint appear at most two literals.

## 2.4 Query Answering in the Presence of Constraints

In [27, 28, 92, 93] it is proposed a framework for data integration that allows the specification of a general form of integrity constraints over the global schema, and it is defined a semantics for data integration in the presence of incomplete and inconsistent information sources. Moreover a method for query processing under the above semantics, when key and foreign key constraints are defined upon the global schema, is proposed.

Formally, the data integration system  $\mathcal{I}$  is a triple  $\langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$  where  $\mathcal{G}$  is the global schema,  $\mathcal{S}$  is the source schema and  $\mathcal{M}_{\mathcal{G},\mathcal{S}}$  is the mapping between  $\mathcal{G}$  and  $\mathcal{S}$ . More specifically, the *global schema* is expressed in the relational model with both key and foreign key constraints, the *source schema* is expressed in the relational model without integrity constraints, and the mapping is defined between the global and source schema, that is each relation in  $\mathcal{G}$  is associated with a view, that is a query over the sources.

*Example 2.5.* An example of a data integration system, reported in [27], is  $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$  where  $\mathcal{G}$  is constituted by the following relation symbols:

$$\begin{aligned} &student(Scode, Sname, Scity) \\ &university(Ucode, Uname) \\ &enrolled(Scode, Ucode) \end{aligned}$$

and the constraints:

$$\begin{aligned}
 \text{key}(\text{student}) &= \text{Scode} \\
 \text{key}(\text{university}) &= \text{Ucode} \\
 \text{key}(\text{enrolled}) &= \text{Scode}, \text{Ucode} \\
 \text{enrolled}[\text{Scode}] &\subseteq \text{student}[\text{Scode}] \\
 \text{enrolled}[\text{Ucode}] &\subseteq \text{university}[\text{Ucode}]
 \end{aligned}$$

$\mathcal{S}$  consists of the three sources:  $s_1$ , of arity 4, containing information about students with their code, name, city and date of birth;  $s_2$ , of arity 2, containing codes and names of universities; and  $s_3$ , of arity 2, containing information about enrollment of students in universities.

The mapping  $\mathcal{M}_{\mathcal{G},\mathcal{S}}$  is defined by:

$$\begin{aligned}
 \rho(\text{student}) &= \text{student}(X, Y, Z) \leftarrow s_1(X, Y, Z, W) \\
 \rho(\text{university}) &= \text{university}(X, Y) \leftarrow s_2(X, Y) \\
 \rho(\text{enrolled}) &= \text{enrolled}(X, Y) \leftarrow s_3(X, Y)
 \end{aligned}$$

□

The semantics of a data integration system is given by considering a source database  $\mathcal{D}$  for  $\mathcal{I}$ , that is a database for the source schema  $\mathcal{S}$  containing a relation  $r^{\mathcal{D}}$  for each source  $r$  in  $\mathcal{S}$ . Any database for  $\mathcal{G}$ , say  $\mathcal{B}$ , is a *global database* for  $\mathcal{I}$ , and it is said to be *legal* with respect to  $\mathcal{D}$  if it:

- satisfies the integrity constraints defined on  $\mathcal{G}$ ;
- satisfies the mapping with respect to  $\mathcal{D}$ , that is for each relation  $r$  in  $\mathcal{G}$ , the set of tuples  $r^{\mathcal{B}}$  that  $\mathcal{B}$  assigns to  $r$  contains of the set of tuples  $\rho(r)^{\mathcal{D}}$  computed by the associated query  $\rho(r)$  over  $\mathcal{D}$ :

$$\rho(r)^{\mathcal{D}} \subseteq r^{\mathcal{B}}$$

Note that each view is considered *sound*, that is the data provided by the sources are not necessary complete. It is possible to formulate other assumption on views [2], in particular a view may be *complete*, that is for each view in  $\mathcal{G}$  it is  $\rho(r)^{\mathcal{D}} \supseteq r^{\mathcal{B}}$  or *exact*, that is for each view in  $\mathcal{G}$  it is  $\rho(r)^{\mathcal{D}} = r^{\mathcal{B}}$ .

**Definition 2.6.** Given a source database  $\mathcal{D}$  for  $\mathcal{I}$ , the semantics of  $\mathcal{I}$  with respect to  $\mathcal{D}$ , denoted by  $\text{sem}^{\mathcal{D}}(\mathcal{I}, \mathcal{D})$  is the set of database defined as follows:

$$\text{sem}^{\mathcal{D}}(\mathcal{I}, \mathcal{D}) = \{\mathcal{B} \mid \mathcal{B} \text{ is a legal global database for } \mathcal{I}, \text{ with respect to } \mathcal{D}\}$$

If  $\text{sem}^{\mathcal{D}}(\mathcal{I}, \mathcal{D}) \neq \emptyset$ , then  $\mathcal{I}$  is said to be *consistent with respect to*  $\mathcal{D}$ . □

Thus, the semantics of a data integration system is given in terms of a set of databases.

A query  $q$  to a data integration system  $\mathcal{I}$  is a conjunctive query over the global schema, whose atoms have symbols in  $\mathcal{G}$  as predicates. A tuple  $(c_1, \dots, c_n)$  is considered an answer to the query only if it is a *certain* answer, that is if it satisfies the

query in every database that belongs to the semantics of the data integration system. More formally, a *certain answer* of a query  $q$  with arity  $n$  with respect to  $\mathcal{I}$  and  $\mathcal{D}$  is the set:

$$q^{\mathcal{I}, \mathcal{D}} = \{(c_1, \dots, c_n) \mid \text{for each } \mathcal{DB} \in \text{sem}(\mathcal{I}, \mathcal{D}), (c_1, \dots, c_n) \in q^{\mathcal{DB}}\}$$

where  $q^{\mathcal{DB}}$  denotes the result of evaluating  $q$  in the database  $\mathcal{DB}$ .

The *retrieved global database*, denoted by  $\text{ret}(\mathcal{I}, \mathcal{D})$  is obtained by computing, for each relation  $r$  of the global schema, the relation  $r^{\mathcal{D}}$ ; this is done by evaluating the query  $\rho(r)$  over the source database  $\mathcal{D}$ . Note that the *retrieved global database* satisfies all the key constraints in  $\mathcal{G}$ , as it is assumed that  $\rho(r)$  does not violate the key constraints, thus if  $\text{ret}(\mathcal{I}, \mathcal{D})$  also satisfies the foreign key constraints then the answer to a query  $q$  can be done by simply evaluating it over  $\text{ret}(\mathcal{I}, \mathcal{D})$ . If it is the case that  $\text{ret}(\mathcal{I}, \mathcal{D})$  violates the foreign key constraints then tuples have to be added to the relations of the global schema in order to satisfy them. Obviously in general there are an infinite number of legal databases that are coherent with the retrieved global database, even if it is shown that there exists one, the *canonical database*, denoted by  $\text{can}(\mathcal{I}, \mathcal{D})$ , that represents all the legal databases that are coherent with the retrieved global database.

Thus, formally the answer to a query  $q$  can be given by evaluating it on  $\text{can}(\mathcal{I}, \mathcal{D})$ . Anyhow, the computation of the canonical database is impractical as, generally, the number of databases can be infinite, thus in [27] it is defined an algorithm that computes the certain answers of a conjunctive query  $q$  without actually building  $\text{can}(\mathcal{I}, \mathcal{D})$ .

The algorithm transforms the original query  $q$  into a new query, called the *expansion of  $q$  with respect to  $\mathcal{G}$* , denoted as  $\text{exp}_{\mathcal{G}}(q)$ , over the global schema, so that the answer  $\text{exp}_{\mathcal{G}}(q)$  over the (virtual) retrieved global database is equal to the answer to  $q$  over the canonical database, that is  $\text{exp}_{\mathcal{G}}(q)$  is independent of the source database  $\mathcal{D}$ .

Roughly speaking, the algorithm is based on the idea of expressing foreign key constraints in terms of rules of a logic program  $\mathcal{P}_{\mathcal{G}}$  with functional symbols (used as Skolem functions).

In order to build the program  $\mathcal{P}_{\mathcal{G}}$ :

- A new relation  $r'$ , called primed relation, is added for each relation  $r$  in  $\mathcal{G}$ .

$$r'(X_1, \dots, X_n) \leftarrow r(X_1, \dots, X_n)$$

- for each foreign key  $r_1[A] \subseteq r_2[B]$  in  $\mathcal{G}$  where  $A$  and  $B$  are sets of attributes and  $B$  is a foreign key for  $r_2$  the rule:

$$r'_2(X_1, \dots, X_h, f_{h+1}(X_1, \dots, X_h), \dots, f_n(X_1, \dots, X_h)) \leftarrow r'_1(X_1, \dots, X_h, \dots, X_n)$$

is added, where  $f_i$  are Skolem functions and it is assumed, for simplicity, that the first  $h$  attributes are involved in the foreign key.

The program  $\mathcal{P}_{\mathcal{G}}$  is then used to generate the query ( $\text{exp}_{\mathcal{G}}(q)$ ) associated to  $q$ . In particular  $\mathcal{P}_{\mathcal{G}}$  is used to generate the *partial evaluation tree* of the query  $q$ , whose non-empty leaves constitute the reformulation ( $\text{exp}_{\mathcal{G}}(q)$ ) of the query  $q$ .

*Example 2.7.* Suppose the global schema  $\mathcal{G}$  of a data integration system consists of the following three relations:

$$\begin{aligned} & person(Pcode, Age, CityofBirth) \\ & student(Scode, Univerisity) \\ & enrolled(Scode, Ucode) \end{aligned}$$

with the constraints:

$$\begin{aligned} key(person) &= Pcode \\ key(student) &= Scode \\ key(city) &= Name \\ person[CityofBirth] &\subseteq city[Name] \\ city[Major] &\subseteq person[Pcode] \\ student[SCode] &\subseteq person[Pcode] \end{aligned}$$

The logic program  $\mathcal{P}_{\mathcal{G}}$  uses the predicate *person*, of arity 3, *student*, with arity 1 and *city* with arity 2 and constitutes the following program:

$$\begin{aligned} person'(X, Y, Z) &\leftarrow person(X, Y, Z) \\ student'(X, Y) &\leftarrow student(X, Y) \\ city'(X, Y) &\leftarrow city(X, Y) \\ city'(X, f_1(X)) &\leftarrow person'(Y, Z, X) \\ person'(Y, f_2(Y), f_3(Y)) &\leftarrow city'(X, Y) \\ person'(Y, f_4(X), f_5(X)) &\leftarrow student'(X, Y) \end{aligned}$$

Suppose we have the query  $q$ :

$$q(X) \leftarrow person(X, Y, Z)$$

The non-empty leaves of the partial evaluation tree of  $q$  provide the following expansion  $q' = exp_{\mathcal{G}}(q)$  of the query:

$$\begin{aligned} q'(X) &\leftarrow person(X, Y, Z) \\ q'(X) &\leftarrow student(X, W_1) \\ q'(W_2) &\leftarrow city(Z, W_2) \end{aligned}$$

Thus the expanded query searches for codes of persons not only in the relation *person*, but also in *student* and *city*, where, due to the integrity constraints, it is known that codes of persons are stored.  $\square$

The above approach is further extended in [92] where the query answer problem in the same setting, but under a loosely-sound semantics of the mapping is investigated. The difference with respect to the previous case can be seen in a situation in which there is no global database that is both coherent with  $\mathcal{G}$  and satisfies the mapping with respect to  $\mathcal{D}$ . In this case  $ret(\mathcal{I}, \mathcal{D})$  violates the constraints in  $\mathcal{G}$ , that is there exists  $r \in \mathcal{G}$  and  $t_1, t_2 \in ret(\mathcal{I}, \mathcal{D})$  such that  $key(r) = X$ ,  $t_1[X] = t_2[X]$ , and  $t_1 \neq t_2$ .

Under the strictly-sound semantics this means that there are no legal databases for  $\mathcal{I}$  with respect to  $\mathcal{D}$ .

In order to avoid this problem it is defined a loosely-sound semantics that allows to always have a coherent database by restricting the set of tuples to those satisfying the constraints. The semantics allows the elimination of tuples from  $ret(\mathcal{I}, \mathcal{D})$ , in particular it implies that the legal databases are the ones that are “as sound as possible”, thus it considers only databases coherent with the constraints that “minimize” the elimination of tuples from  $ret(\mathcal{I}, \mathcal{D})$ .

The method for computing a certain answer identifies the legal databases with respect to  $\mathcal{I}'$ , obtained from  $\mathcal{I}$  by eliminating all the foreign key constraints in  $\mathcal{G}$ . Obviously, each of such databases  $\mathcal{B}$  is contained in  $ret(\mathcal{I}, \mathcal{D})$ . Then the legal databases are used in the query reformulation technique for the strictly-sound semantics previously illustrated.

## 2.5 Complete Answers from Incomplete Databases

In [94] the problem of answering queries from databases that may be incomplete is considered. A database is *incomplete* or *partial* if tuples in each relation are only a subset of the tuples that *should* be in the relation and, generally, only a part of each relation is known to be complete. Formally, this situation can be modeled as having two sets of relations, the *virtual* and the *available* relations. The virtual relations are  $\bar{R} = R_1, \dots, R_n$  while the available relations are  $\bar{R}' = R'_1, \dots, R'_n$  and for every  $i \in \{1..n\}$  the extension of the available relation  $R'_i$  contains a *subset* of the tuples in the extension of the virtual relation  $R_i$ .

The important question addressed in [94] is the *answer completeness* problem, that is deciding whether an answer to a given query is guaranteed to be complete even if the database is incomplete or in other words if the answer is guaranteed to contain all the tuples we would have obtained by evaluating the query over the virtual relations. Clearly, if it is known that  $R'_i \subseteq R_i$  for each  $i$ ,  $1 \leq i \leq n$ , then the answer to the query may be incomplete; however, it is often the case that an available relation, say  $R'_i$  has the property of being *partially complete*, that is some parts of  $R'_i$  are identical to  $R_i$ . The local completeness property guarantees that if the answer to the query just depends on the complete portion it is guaranteed to be complete.

Local completeness for a relation  $R'$  is specified by a constraint on the tuples of  $R$  that are *guaranteed* to be in  $R'$ .

More formally:

**Definition 2.8. (Constraints)** Let  $R$  be a relation of arity  $n$ , and  $X_1, \dots, X_n$  be variables standing for its attributes. A constraint  $C$  on the relation  $R$  is a conjunction of atoms that includes constants, variables from  $X_1, \dots, X_n$  and other variables. The relations used in  $C$  can be either database relations or comparison predicates, but not  $R$  itself. A tuple  $(a_1, \dots, a_n)$  satisfies  $C$  with respect to a database instance

if the conjunction resulting from substituting  $a_i$  for  $X_i$  in  $C$  is satisfied in  $\mathcal{I}$ . The complement of  $C$  is denoted by  $\neg C$ .  $\square$

**Definition 2.9. (Local Completeness)** Let  $C$  be a constraint on the relation  $R$ . A database instance  $\mathcal{I}$  that includes the relations  $R$  and  $R'$  is said to satisfy the local-completeness statement  $LC(R', R, C)$  if  $r'$  contains all the tuples of  $R$  that satisfy  $C$ , that is if the results of the following two queries are identical over  $\mathcal{I}$ :

$$\begin{aligned} q_1(X_1, \dots, X_n) &\leftarrow R(X_1, \dots, X_n), C \\ q_2(X_1, \dots, X_n) &\leftarrow R'(X_1, \dots, X_n), C \end{aligned}$$

$\square$

The solution to the answer-completeness problem is given by showing that this problem is equivalent to the one of detecting the independence of a query from an insertion update, that is the problem of determining whether the answer to a query changes as a result of an insertion to the database. In particular let  $Q$  be a union of conjunctive queries over the virtual relations  $\bar{R}$  and comparison predicates, and let  $\Gamma$  be a set of local completeness statements of the form  $LC(R'_j, R_j, C_j)$ , where  $R'_j \in \bar{R}'$  and  $R_j \in \bar{R}$ . The query  $Q$  is answer-complete w.r.t  $\Gamma$  if and only if  $In^+(Q, (R_j, \neg C_j))$  holds for every statement in  $\Gamma$ .  $In^+(Q, (R_j, \neg C_j))$  states that the query  $Q$  is independent from the insertion update  $(R_j, \neg C_j)$ , that is for any database instance  $\mathcal{I}$  and any database instance  $\mathcal{I}'$  that results from  $\mathcal{I}$  by adding to  $R$  some tuples that satisfy  $\neg C_j$ ,  $Q(\mathcal{I}) = Q(\mathcal{I}')$ .

**Theorem 2.10.** Let  $Q$  be a union of conjunctive queries over the virtual relations  $\bar{R}$  and comparison predicates, and let  $\Gamma$  be a set of local completeness statements of the form  $LC(R'_j, R_j, C_j)$ , where  $R'_j \in \bar{R}'$  and  $R_j \in \bar{R}$ . The query  $Q$  is answer-complete w.r.t  $\Gamma$  if and only if  $In^+(Q, (R_j, \neg C_j))$  holds for every statement in  $\Gamma$ .

where  $In^+(Q, (R_j, \neg C_j))$  states the query  $Q$  is independent from the insertion update  $(R_j, \neg C_j)$ , that is for any database instance  $\mathcal{I}$  and any database instance  $\mathcal{I}'$  that results from  $\mathcal{I}$  by adding to  $R$  some tuples that satisfy  $\neg C_j$ ,  $Q(\mathcal{I}) = Q(\mathcal{I}')$ .  $\square$

With the previous theorem the problem of detecting independence can be solved by using one of the algorithms studied in the literature.

The equivalence problem is undecidable for recursive queries [132], while it is decidable in the following cases:

- if each of the  $C_j$ 's contains only arguments of  $R_j$  or constants, or
- if the head of  $Q$  contains all the variables of the body of  $Q$ , and neither the  $C_j$ 's or  $Q$  use the comparison predicates.

Generally, the problem of deciding answer-completeness is in  $\Pi_2^P$ . The best known algorithm for the independence problem and therefore for the answer completeness problem is exponential, even if it has been shown that if updates are described using a conjunction of comparison predicates the independence problem can be decided in polynomial time.

## 2.6 Condensed Representation of Database Repairs for Consistent Query Answering

In [145] a general framework for repairing databases is proposed. In particular the author stressed that an inconsistent database can be repaired without deleting tuples (tuple-based approach), but using a finer repair primitive consisting in correcting faulty values within the tuples, without actually deleting them (value-based approach).

*Example 2.11.* Suppose to have the following set of tuples reporting the dioxin levels in food samples:

<i>Sample</i>	<i>SampleDate</i>	<i>Food</i>	<i>AnalysisDate</i>	<i>Lab</i>	<i>DioxinLevel</i>
110	17 Jan 2002	poultry	18 Jan 2002	ICI	normal
220	17 Jan 2002	poultry	18 Jan 2002	ICB	alarming
330	18 Jan 2002	beef	18 Jan 2002	ICB	normal

*Dioxin Database*

and the constraints :

$$\forall S, D1, F, D2, L, D(Dioxin(S, D1, F, D2, L, D) \supset D1 \leq D2)$$

that imposes that the date of analyzing a given sample cannot precede the date the sample was taken.

The first tuple in the Dioxin Database says that the sample 110 was taken on 17 Jan 2002 and analyzed the day after at the *ICI* lab, and that the dioxin level of this sample was normal. While sample 110 respects the constraint, sample 220 violates it. An inconsistency is present in the database and the author claims to *clean* it in a way that avoids deleting the entire tuple, that is acting at the attribute level and not at the tuple level.  $\square$

Given an inconsistent database a consistent answer can be obtained by leaving the database in its inconsistent state, and by propagating the consistent portion of the database in the answer, that is the set of tuples matching the query and satisfying the constraints. Since the repair work is deferred until query time this approach is called *late-repairing*. Given a satisfiable set of constraints  $\eta$ , that is a set of finite constraints, and a relation  $I$ , apply a database transformation  $h_\eta$  such that for every query  $Q$ ,  $Q(h_\eta(I))$  yields exactly the consistent answer to  $Q$  on input  $I$  and  $\eta$ .

Observe that  $h_\eta(I)$  is not necessarily a repair for  $I$  and  $\eta$ , and can be thought as a *condensed representation* of all possible repairs for  $I$  and  $\eta$  that is sufficient for consistent query answering. The practical intuition is that an inconsistent database  $I$  is firstly transformed through  $h_\eta$  in such a way that the subsequent queries on



the transformed database retrieve exactly the consistent answer. Since database is modified prior to query execution, this approach is called *early-repairing*.

### General Repair Framework

Before formally introducing the framework let's give some preliminaries. The framework focuses on a unirelational database, and the set of constraints, denoted  $\eta$ , are expressed in a first-order (*FO*) language using a simple  $n$ -ary predicate symbol. A substitution  $\theta$  is a set of pairs  $\{X_1/t_1, \dots, X_k/t_k\}$  where  $X_1, \dots, X_k$  are distinct variables,  $t_1, \dots, t_k$  are terms and no variable  $X_i$  appears in any term  $t_j$ . The application of a substitution  $\theta$  to a set of literals  $S$ , written  $S\theta$ , is the set of literals derived from  $S$  by the simultaneous replacing of all the variables appearing in  $\theta$  with the associated terms. A substitution  $\theta$  is a unifier for a set of literals  $S$  if  $S\theta$  is a singleton. We say that a set of literals  $S$  *unify* if there exists a unifier  $\theta$  for  $S$ . A unifier  $\theta$  for  $S$  is called a *most general unifier (mgu)* for  $S$  if, for each unifier  $\sigma$  of  $S$ ,  $\sigma$  is an instance of  $\theta$ , i.e. there is a substitution  $\delta$  such that  $\sigma = \theta\delta$ . A *tableau* is a relation that can contain variables. A tableau  $T$  is said to  $\theta$ -*subsume* a tableau  $S$ , here denoted  $T \succeq S$ , if there exists a substitution  $\theta$  such that  $\theta(S) \subseteq T$ . The  $\theta$ -*subsumption*, commonly used between clauses, is here used between tableaux representing the negation of a clause: the tableau  $\{t_1, \dots, t_m\}$ , can be treated as  $\exists^*(t_1 \wedge \dots \wedge t_m)$ , that is as the negation of the clause  $\forall^*(t_1 \wedge \dots \wedge t_m)$ . Clearly,  $T \supseteq S$  implies  $T \succeq S$ ; hence  $\theta$ -*subsumption* weakens the order  $\supseteq$ . If  $G$  is a tableau, then  $grad(T)$  denotes the smallest relation that contains every ground tuple of  $T$ . A valuation is a mapping from variables to constants, extended to be the identity on constants, a substitution is a mapping from variables to symbols, extended to be the identity on constants. Valuation and substitution are extended to tuples and tableaux in a natural way. We write  $id$  for the identity function on symbol; and  $id_p = q$ , where  $p$  and  $q$  are not two distinct constants, for a substitution that identifies  $p$  and  $q$  and that is the identity otherwise. That is if  $p$  is a variable and  $q$  a constant, then  $id_{p=q} = \{p/q\}$ . If  $p$  and  $q$  are variables, then  $id_p = q$  can be either  $\{p/q\}$  or  $\{q/p\}$ . Given two tableaux  $T$  and  $S$ , of the same given arity, we write  $S \succeq T$  iff there exists a substitution  $\theta$  such that  $\theta(T) \subseteq S$ . We write  $S \sim T$  iff  $S \succeq T$  and  $T \succeq S$ ; we write  $S \succ T$  iff  $S \succeq T$  and it  $S \sim T$  does not hold. A relation  $F$  (a tableau in this context) *satisfies*  $\eta$  if there exists a relation  $J \succeq F$  such that  $J \models \eta$ .

*Fixing* (or *repairing*) a relation  $I$  with respect to a set  $\eta$  of integrity constraints means modifying  $I$  in order to bring it in accordance with  $\eta$ , by ensuring the *minimal change* principle, that is the result of fixing has to be as close as possible to the initial relation. In particular fixing a relation is an operation consisting of *downfixing* followed by *upfixing*.

*Downfixing* means that we pass from  $I$  to  $F$ , called *fix*, so that  $I \succeq F$  and  $F$  satisfies  $\eta$ . *Upfixing* means that we subsequently pass from  $F$  to a relation  $M \succeq F$  such that  $M \models \eta$ , where  $M$  is called *mend*. In fixing a relation it is required that the result of fixing is as close as possible to the initial relation.

In this framework the minimal change principle is settled by using the *maximal content preservation criterion*: downfixing retains as much as possible from the original relation, and upfixing consists of a minimal completion:  $F$  should be such that there exists no  $F'$  that also subsatisfies  $\eta$  and such that  $I \succeq F' \succ F$ , that is  $F'$  is *closer* to  $I$  than  $F$ . Next, for a given  $F$ ,  $M$  should be such that there exists no  $M'$  such that  $M \succ M' \succeq F$  and  $M' \models \eta$ . This criteria only relies on the order  $\succeq$ .

For the order  $\succeq$  the  $\supseteq$  or the  $\theta$ -*subsumption* could be chosen. Anyhow, the author points out that both prove to be inadequate. In particular  $\supseteq$  is too *strong* for repairing as it does not allow to differentiate between tuples that agree on most attributes and tuples that disagree on all attributes: the tuples are simply treated as unequal in both cases, thus the repairing is tuple-based. On the other side  $\succeq$  is too weak for downfixing as it can produce mends with spurious tuples.

Therefore, the author claims downfixing has to be based on a relation, denoted  $\sqsubseteq$ , in between  $\supseteq$  and  $\succeq$ . More formally, given two tableaux  $T$  and  $S$ , of the same given arity,  $S \sqsubseteq T$  iff there exists a substitution  $\theta$  such that  $\theta(T) \subseteq S$  and  $|\theta(T)| \subseteq |T|$ . The latter condition ensures that  $\theta$  does not identify distinct tuples of  $T$ .

Related to the chosen order,  $\leq$ , fix and mend are defined as follows. Given a relation  $I$ , of arity  $n$ , and a set of constraints  $\eta$ :

- a *fix* for  $I$  and  $\eta$  is a tableau  $F$  such that  $I \sqsubseteq F$ ,  $F$  subsatisfies  $\eta$ , and for every tableau  $F'$  if  $I \supseteq F' \succ F$ , then  $F'$  does not subsatisfy  $\eta$ .
- a *mend* for  $I$  and  $S$  is a relation  $M$  with  $M \models \eta$  such that there exists a fix  $F$  for  $I$  and  $\eta$  satisfying : (i)  $M \succeq F$  and (ii) for every relation  $M'$ , if  $M \succ M' \succeq F$ , then  $M'$  does not satisfy  $\eta$ .

Note that the requirement  $I \sqsubseteq F$  in the above definition implies the existence of a substitution  $\theta$  such that  $\theta(F) \subseteq I$  and  $|\theta(F)| = |F|$ , thus for a given tuple  $t \in I$  there can be at most one repairing tuple  $t' \in F$  such that  $\theta(t') = t$ .

### Trustable Query Answers

Obviously, for a given relation and a given set of constraints, the number of mends is generally infinite. Thus the author investigates the problem of querying these mends in order to obtain a consistent answer, here called *trustable answer*, that is answer satisfying the set of constraints.

More formally given a unirelational database consisting of a relation  $I$ , of arity  $n$ , and a set of constraints  $\eta$  and a query  $q$ , the ground tuple  $t$  is a trustable answer to  $q$  on input  $\eta$  iff  $t \in q(M)$  for every mend  $M$  for  $I$  and  $\eta$ .

*Example 2.12.* Continuing the previous example. Let us consider the query:

$$\text{Answer}(S) \leftarrow \text{Dioxin}(S, D1, F, D2, L, \text{"alarming"})$$

asking for samples with an alarming dioxin level. The identification 220 is a trustable answer, but it is not a trustable answer for the query asking for a sample date of 17 Jan 2002. In fact, many mends show a different sample date for the sample 220.

A class  $Q$  of queries is *early-repairable* with respect to a class of constraints  $\mathbf{C}$ , if for every satisfiable set of constraints  $\eta \in \mathbf{C}$  and for every relation  $I$ , there exists a computable relation  $I'$  such that for every query  $q \in Q$ ,  $q(I')$  is exactly the set of trustable answers to  $q$  on input  $I$  and  $\eta$ . After formally defining the trustable answer the author focuses on the classes of queries and constraints for which trustable answers can be effectively computed, examining conjunctive queries and full dependencies.

### Tableau Queries and Full Dependencies

A *tableau query* is a pair  $(B, h)$  where  $B$  is a tableau and  $h$  is a tuple (called *summary*) such that every variable in  $h$  also occurs in  $B$ ;  $B$  and  $h$  need not have the same arity. Let  $\tau = (B, h)$  be a tableau query, and  $T$  a tableau of the same arity as  $B$ . A tuple  $t$  is an *answer* to  $\tau$  on input  $T$  iff there exists a substitution  $\theta$  for the variables in  $B$  such that  $\theta(B) \subseteq T$  and  $\theta(h) = t$ . The set of all answers to  $\tau$  on input  $T$  is denoted  $\tau(T)$ .

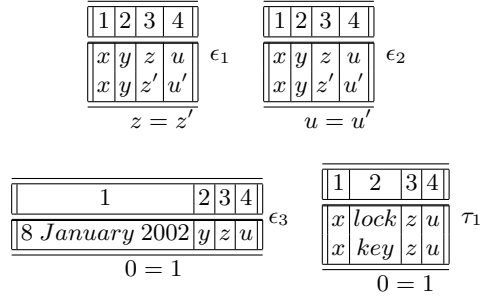
A *full dependency* is either a full *tuple-generating dependency* (*ftgd*) or a full *equality-generating dependency* (*fegd*). A *ftgd* takes the form of a conjunctive query  $(B, h)$  where  $B$  and  $h$  have the same arity. The *ftgd*  $\tau = (B, h)$  is satisfied by a tableau  $T$ , denoted  $T \models \tau$ , iff  $T \cup \tau(T) \sim T$ . A *fegd* is of the form  $(B, p = q)$  where  $B$  is a tableau and  $p$  and  $q$  are symbols such that every variable in  $\{p, q\}$  also occurs in  $B$ . The *fegd*  $\epsilon = (B, p = q)$  is satisfied by a tableau  $T$ , denoted  $T \models \epsilon$ , iff for every substitution  $\theta$ , if  $\theta(B) \subseteq T$  then  $\theta(p), \theta(q)$  are not two distinct constants and  $T \sim id_{\theta(p)=\theta(q)}(T)$ .

*Example 2.13.* Consider a relation *Manufacture* with four attributes denoting *date*, *product*, *color* and *quantity* respectively. For example a tuple  $(12 \text{ Jan } 2002, \text{lock}, \text{green}, 1000)$  means that *1000 green locks were manufactured on 12 Jan 2002*. The production line is subject to the constraints reported in Figure 2.13

In particular, the *fegd*'s  $\epsilon_1$  and  $\epsilon_2$  express that the date and the product uniquely identify tuples in *Manufacture*.  $\epsilon_2$  captures the fact that 8 Jan 2002 was a day of strike, on which no products were manufactured (0 and 1 can be replaced by any two distinct constants). Finally the *ftgd*  $\tau_1$  expresses that each production of a lock involves the simultaneous production of a key in the same color.

The author shows that given two tableaux  $T$  and  $S$  and a set of full dependencies  $\eta$ , if  $T \sim S$  and  $T \models \eta$ , then  $S \models \eta$ .

It is known that every finite set  $S$  of tableaux has a greatest lower bound under  $\succeq$ . More formally a tableau  $L$  is a *lower bound* of a finite set  $S$  of tableaux iff for each  $T \in S$ ,  $T \succeq L$ . A lower bound  $G$  of  $S$  is called the *greatest lower bound* (*glb*) of  $S$  iff  $G \succeq L$  for every lower bound  $L$  of  $S$ .



The construction of *glb* and tableau query commute up to  $\sim$ . In fact, given two tableaux  $T, S$ , a tableau query  $\tau = (B, h)$ , a *glb*  $\{T, S\}$  and a *glb* of  $\{\tau(T), \tau(S)\}$ , then  $\tau(G) \sim F$ .

### Chasing Fixes

The *chase*, originally introduced for deciding logical implication is used for repairing databases. In particular, some results are generalized to tableaux that can contain constants, need not be typed and in these equality is replaced by  $\sim$ .

An artificial top element, denoted  $\square$ , is introduced to the semi-order  $\langle T, \square \rangle$ . Let  $T \neq \square$  and  $S$  be tableaux and  $\eta$  a set of full dependencies. We write  $T \models_{\eta} S$  if  $S$  can be obtained from  $T$  by a single application of one of the following chase rules:

- If  $\tau = (B, h)$  is a *ftgd* on  $\eta$ , then  $T \models_{\eta} T \cup \tau(T)$ .
- Let  $(B, p = q)$  be a *fgd* of  $\eta$ , and  $\theta$  a substitution such that  $\theta(B) \subseteq T$ .
- If  $\theta(p)$  and  $\theta(q)$  are two distinct constants, then  $T \models_{\eta} \square$ ; otherwise,  $T \models_{\eta} id_{\theta(p)=\theta(q)}(T)$ .

A chase of  $T$  by  $S$  is a *maximal* (with respect to length) sequence  $T = T_0, T_1, \dots, T_n$  of tableaux such that for every  $i \in \{1, \dots, n\}$ ,  $T_{i-1} \models_{\eta} T_i$  and  $T_i \neq T_{i-1}$ .

Requiring that chases be maximal tacitly assumes that chases are finite.

Given a tableau  $F \neq \square$  and a set of full dependencies, then

- If  $T$  is a tableau in a chase of  $F$  by  $\eta$ , then  $T \succeq F$ .
- Each chase of  $F$  by  $S$  is finite.
- If  $T \neq \square$  is the last element of a chase of  $F$  by  $\eta$ , then  $T \models_{\eta}$ .
- If  $T \neq \square$  is the last element of a chase of  $F$  by  $\eta$ , and  $\theta$  is a valuation mapping distinct variables to new distinct constants not occurring elsewhere, then  $\theta(T) \models_{\eta}$ .

The author shows that given a set of full dependencies  $\eta$  and a tableau  $F \neq \square$ , then  $F$  subsatisfies  $\eta$  iff  $chase(F, \eta) \neq \square$ . Thus a set of full dependencies  $\eta$  is satisfiable iff  $chase(\emptyset, S) \neq \square$ .

*Example 2.14.* Let's continue the previous Example. The Figure 5.1 shows a *Manufacture* relation together with fixes and chase results. The integrity constraints are violated: no items can have been produced on *8 Jan 2002*, and the production of 100 blue locks must entail 100 blue keys. Moreover red and blue keys cannot have been manufactured on the same day.

Sample	SampleDate	Food	AnalysisDate	Lab	DioxinLevel
110	17Jan2002	poultry	18Jan2002	ICI	normal
220	17Jan2002	poultry	18Jan2002	ICB	alarming
330	18Jan2002	beef	18Jan2002	ICB	normal

*Manufacture*

1	2	3	4	$F_1$	1	2	3	4	$chase(F_1, \eta)$
$x$	lock	blue	110		$x$	lock	blue	110	
$x$	key	$z$	110		$x$	key	blue	110	
1	2	3	4	$F_2$	1	2	3	4	$chase(F_2, \eta)$
$x$	lock	$z$	110		$x$	lock	red	110	
$x$	key	red	110		$x$	key	red	110	
1	2	3	4	$F_3$	1	2	3	4	$chase(F_3, \eta)$
$x$	lock	blue	110		$x$	key	red	110	
$x$	key	red	110		$x$	key	blue	110	
1	2	3	4	$F_4$	1	2	3	4	$chase(F_4, \eta)$
$x$	lock	blue	110		$x$	$y$	red	110	
$x$	$y$	red	110		$x$	key	blue	110	
1	2	3	4	$F_5$	$chase(F_5, \eta) \sim F_5$				
$x$	$y$	blue	110						
$x$	key	red	110						

**Fig. 2.1.** Manufactures Databases with fixes and chase results

$F_1$  and  $F_2$  assume that the date of *8 Jan 2002* and either color (red or blue) were mistaken.  $F_4$  and  $F_5$  assume that the date of *8 Jan 2002* and either product (key or lock) were mistaken. Finally,  $F_3$  assumes that the date of *8 Jan 2002* should be replaced by different dates in either tuple of *Manufacture*. It is easy to verify that any other fix is equivalent under  $\sim$  to one of the five fixes shown.

The formal definition of *trustable tableau* is as follows: let  $\mathbf{F}$  be a minimal set of tableaux (with respect to  $\subseteq$ ) such that for every fix  $F$  for  $I$  and  $\eta$ , there exists

some tableau  $F'$  such that  $F' \sim F$ . Let  $\mathbf{S}$  be a minimal (with respect to  $\subseteq$ ) set of tableaux such that for every  $F \in \mathbf{F}$ , there exists some tableau  $T \in \mathbf{S}$  such that  $T \in \text{chase}(F, \eta)$ . Let  $G$  be a *glb* of  $\mathbf{S}$ , then  $G$  is called a trustable tableau for  $I$  and  $S$ .

Computation of trustable tableau is shown to be computable for unirelation databases with a set of full dependencies. The computation is quite complex as it involves solving *NP*-complete problems, like deciding the  $\theta$ -*subsumption* for determining the fixing.

## 2.7 Using Views

Recent research on databases has been concerned with the problem of answering queries when only materialized views are available as base relations [35, 36, 47, 95, 96, 122, 141]. This problem is becoming more and more important in many areas such as query optimization, database design, data integration, data warehouse, mobile computing and others. The many applications of the problem of answering queries using views has spurred a flurry of research, which also led to the design and implementation of several commercial system.

Informally speaking, given a general query  $Q$  over a database schema, and a set of view definitions  $\mathcal{V}_1, \dots, \mathcal{V}_n$  over the same schema, the problems are the following: is it possible to answer the query  $Q$  using *only* the answer to the views  $\mathcal{V}_1, \dots, \mathcal{V}_n$ ; and what is the maximal set of tuples in the answer of  $Q$  that we can obtain from the views?

The treatment of the problem differs mainly depending on whether it is concerned with query optimization, database design or with data integration. The main distinction is essentially between works on query optimization and maintenance of physical data independence and works concerning data integration. In particular, in the context of query optimization and database design, the focus has been on producing a query execution plan that involves the views, and hence the effort has been on extending query optimizers to accommodate the presence of views; whereas in the context of database integration, the focus has been on translating queries, formulated in terms of the mediated schema into queries formulated in terms of the data sources. Thus the key difference between these two classes of works is the output of the algorithm for answering queries using views. In the former case, given a query  $Q$ , and a set of views  $\mathcal{V}_1, \dots, \mathcal{V}_n$ , the goal of the algorithm is to produce an expression  $Q_1$  that references the views and is either equivalent to or contained in  $Q$ ; whereas in the latter case the algorithm must go further and produce a query execution plan for answering  $Q$  using the views.

A data integration system exposes to the user a mediated schema, which consists of a set of *virtual* relations, in the sense that they are not actually stored anywhere. To be able to answer queries the system must contain a set of *source descriptions*, that describe the contents of the source, the attribute that can be found in the source, and the constraints on the content of the source. One of the preeminent approaches for

specifying source description, adopted in several systems, [49, 91, 97] is to describe the contents of a data source as a view over the mediated schema. This approach allows easily to add new data sources and to specify constraints on the contents of the sources.

Therefore, an interesting problem is to find a rewriting which is *correct*, that is only atoms derived from the original query are derived from the rewritten one, and, possibly, *optimal*, that is it gives the best approximation of the original query.

In this environment the concepts of *query containment* and of *query equivalence* are essential as they provide a semantic basis for comparison between queries and between different reformulations of queries, and can be used to test the correctness of a rewriting. In the following we denote the result of computing the query  $Q$  over the database  $\mathcal{I}$  by  $Q(\mathcal{I})$ .

**Definition 2.15.** A query  $Q_1$  is said to be contained in a query  $Q_2$ , denoted by  $Q_1 \sqsubseteq Q_2$ , if for all database instances  $\mathcal{I}$ , the set of all tuples computed for  $Q_1$  is a subset of those computed for  $Q_2$ , that is  $Q_1(\mathcal{I}) \subseteq Q_2(\mathcal{I})$ . The two queries are said to be equivalent if  $Q_1 \sqsubseteq Q_2$  and  $Q_2 \sqsubseteq Q_1$ .  $\square$

The problem of answering queries using views has also to take into account the different properties of the data sources; as an example it has been shown that if data sources are assumed to be *complete*, that is they include all the tuples that satisfy their definition, then the problem of answering queries using views becomes computationally harder. Intuitively this is due to the fact that when a source is complete it is also possible to infer negative information as a result of a query to the source. This led to a more complex question: given a general query  $Q$  over a database schema, and a set of view definitions  $\mathcal{V}_1, \dots, \mathcal{V}_n$  over the same schema what is the complexity of finding the maximal set of tuples in the answer to  $Q$  from  $\mathcal{V}_1, \dots, \mathcal{V}_n$ . From what previously stated, it is necessary to distinguish between two types of query rewriting: *equivalent rewritings* and *maximally contained rewritings*.

**Definition 2.16.** Equivalent rewriting: Let  $Q_1$  be a query and  $\mathcal{V} = \mathcal{V}_1, \dots, \mathcal{V}_m$  be a set of view definitions. The query  $Q'$  is an equivalent rewriting of  $Q$  using  $\mathcal{V}$  if  $Q'$  refers only to the views in  $\mathcal{V}$  and  $Q'$  is equivalent to  $Q$ .  $\square$

**Definition 2.17.** Maximally contained rewriting: Let  $Q_1$  be a query and  $\mathcal{V} = \mathcal{V}_1, \dots, \mathcal{V}_m$  be a set of view definitions and  $\mathcal{L}$  be a query language. The query  $Q'$  is a maximally contained rewriting of  $Q$  using  $\mathcal{V}$  with respect to  $\mathcal{L}$  if (i)  $Q'$  is a query in  $\mathcal{L}$  that refers only to the views in  $\mathcal{V}$ ; (ii)  $Q'$  is contained in  $Q$  and there is no rewriting  $Q_1 \in \mathcal{L}$ , such that  $Q' \sqsubseteq Q_1 \sqsubseteq Q$  and  $Q_1$  is not equivalent to  $Q'$ .  $\square$

Obviously unlike the case of equivalent rewritings, the maximally contained rewriting may differ depending on the query language, moreover the algorithm for query containment and equivalence provide methods for *testing* whether a candidate rewriting is an equivalent or a contained rewriting, but do not formally provide a solution to the problem of answering queries using views. Another important question stressed

in this context is how to find, given a set of view definitions and their extensions, *all* the possible answers to the query. A maximally contained rewriting does not always provide all the possible answers that can be obtained from the views as the rewriting in this case is specific with respect to a specific language.

Formally, the problem of finding all the answers to a query is formalized by the notion of *certain* answers, introduced in [2] in which it is distinguished the case in which the view extension is complete (closed-world assumption) from the case in which the views may be partial (open-world) [2, 35, 68]. The intuition behind the concept of certain answer is that the extensions of a set of views do not define a unique database instance, thus given the extensions of the views we have only partial information about the real state of the database. A tuple is a certain answer of a query  $Q$  if it is an answer for *any* of the possible database instances that are consistent with the given extensions of the views.

**Definition 2.18.** *Certain answers:* Let  $Q_1$  be a query and  $\mathcal{V}=\mathcal{V}_1, \dots, \mathcal{V}_m$  be a set of view definitions over the database schema  $\mathcal{R}_1, \dots, \mathcal{R}_n$ . Let the sets of tuples  $v_1, \dots, v_n$  be extensions of the views  $\mathcal{V}_1, \dots, \mathcal{V}_m$  respectively. The tuple  $a$  is a certain answer to the query  $Q$  under the closed-world assumption given  $v_1, \dots, v_n$  if  $a \in Q(D)$  for all database instances  $\mathcal{I}$  such that  $\mathcal{V}_i(\mathcal{I}) = v_i$  for every  $i, 1 \leq i \leq m$ . The tuple  $a$  is a certain answer to the query  $Q$  under the open-world assumption given  $v_1, \dots, v_n$  if  $a \in Q(D)$  for all database instances  $\mathcal{I}$  such that  $\mathcal{V}_i(\mathcal{I}) \supseteq v_i$  for every  $i, 1 \leq i \leq m$ .  $\square$

Thus, in order to answer a query using views it is necessary to translate the query, formulated over the mediated schema into a query that directly refers to the schema of the data sources.

Many techniques used to answer queries by means of materialized views have been proposed in the literature, the interested reader can refer to [141] and [48] for a general introduction on the argument, to [49] for the rewriting of positive conjunctive views and to [61] for the rewriting of general views.

Many algorithms for query rewriting have been proposed. In [141] the query posed upon the mediated schema is a conjunctive query ( $CQ$ ), that is a rule with subgoals having  $\mathcal{EDB}$  predicates. A  $CQ$  is applied to the  $\mathcal{EDB}$  relations by considering all possible substitutions of values for the variables in the body. As an example we can consider the query:

$$p(X, Z) \leftarrow a(X, Y), a(Y, Z)$$

where thinking of  $a$  as an “arc” predicate defining a graph, the rule states that “ $p(X, Z)$  is true if there is an arc from the node  $X$  to  $Y$  and an arc from the node  $Y$  to  $Z$ ”, that is there is a path of length two from  $X$  to  $Z$ .

For conjunctive queries the containment first studied by Chandra and Merlin [34] is tested following the approach used in [124]. In more details, to test whether  $Q_1 \subseteq Q_2$  the following steps are performed:



- *freeze* the body of  $Q_1$  by turning each of its subgoals into facts in the database. This can be achieved by replacing each variable in the body by a distinct constant and by treating the resulting subgoals as the only tuples in the database ;
- apply  $Q_2$  to this *canonical* database;
- if the frozen head of  $Q_1$  is derived by  $Q_2$ , then  $Q_1 \subseteq Q_2$ . otherwise not.

If the test is negative then the canonical database is a counterexample to the containment, as surely  $Q_1$  derives its frozen head from this database; whereas if it is positive there is an homomorphism from the variables of  $Q_2$  to the variables of  $Q_1$ . Containment of  $CQ$ 's has been proved to be  $\mathcal{NP}$ -complete in [34] although in [130] it is shown that in the common case where no predicate appears more than twice in the body, then there is a linear-time algorithm for containment.

*Example 2.19.* Let's consider the following two  $CQ$ 's [141]:

$$\begin{aligned} Q_1 &: p(X, Z) \leftarrow a(X, Y), a(Y, Z) \\ Q_2 &: p(X, Z) \leftarrow a(X, U), a(V, Z) \end{aligned}$$

where  $Q_1$  looks for paths of length two, while  $Q_2$  looks only for nodes  $X$  and  $Z$  such that  $X$  has an arc out to elsewhere and  $Z$  has an arc in from elsewhere. The procedure for testing the containment informally described previously supports the intuition that  $Q_1 \subseteq Q_2$ .  $\square$

An important extension of  $CQ$ 's consists in allowing negated subgoals in the body. The effect of applying a  $CQ$  to a database is as before, but in this case when constants are substituted to variables the atoms in the negated subgoals must be false, rather than true (that is the negated atom itself must be true). In the presence of negated subgoals the containment test is slightly more complex, in particular it is  $\Pi_P^2$ -complete.

The query is expressed in terms of the  $\mathcal{EDB}$  predicates. The problem is that of finding a *valid solution*  $\mathbf{S}$  for the query  $Q$ , that is an expression of the query in terms of the views, that is such that it is possible to replace the views in  $\mathbf{S}$  by their definitions, said *expansion*  $E$ , of the query which results to be equivalent to the original query  $Q$ .

*Example 2.20.* Let's suppose to have a single  $\mathcal{EDB}$  predicate  $p(X, Y)$  which states that  $Y$  is parent to  $X$ . Let there be two views, defined as follows[141]:

$$\begin{aligned} v_1(Y, Z) &\leftarrow p(X, Y), p(Y, Z) \\ v_2(X, Z) &\leftarrow p(X, Y), p(Y, Z) \end{aligned}$$

where the first view  $v_1$  produces a subset of the relation for  $p$ , that is those child-parent pairs  $(Y, Z)$  such that the child is also a parent of some individual  $X$ . The second view  $v_2$  models a grandparent relation from the parent relation. The query  $q$

$$q(c) \leftarrow p(0, A), p(A, B), p(B, C)$$

asking for the great grandparents of a particular individual, can be rewritten by the following expansion using only the predicate  $v_1$  and  $v_2$ :

$$s_1(c) \leftarrow v_2(0, D), v_1(D, C)$$

By replacing each of the subgoals of  $s_1$  with the definition of the views we obtain the expansion:

$$e_1(C) \leftarrow p(0, E), p(E, D), p(D, C)$$

Using the containment test in both directions it can be proved that  $e_1 \equiv q$ . Obviously there are other solutions that, when expanded are contained in  $q$ , but are not equivalent to it.  $\square$

From the previous example it results clear that one can only guess potential solutions for a query and then test them using the containment test. However there are theorems that limit the search and show that the problem of expressing a query in terms of views is no worse than  $\mathcal{NP}$ -complete. The idea is that any view used in a solution must serve some function in the query; a view without a function must be deleted from the solution.

A solution  $S$  for a query  $Q$  is *minimal* if :

- $S \subseteq Q$ ;
- there is no solution  $T$  for  $Q$  such that:
  - $S \subseteq T \subseteq Q$  and
  - $T$  has fewer subgoals than  $S$

Both in [95] and in [123] are defined theorems that offer nondeterministic polynomial-time algorithm to find either:

- a single solution equivalent to the query  $Q$
- a set of solution whose union is contained in  $Q$  and that contains any other solution that is contained in  $Q$ .

In each case one searches “only” an exponential number of minimal queries. If what we are looking for is a solution equivalent to  $Q$  then we may stop if we find one; whereas we can conclude there is none if we have searched all solutions and found none.

A technique for answering Datalog queries using views restricted to being positive and conjunctive was presented in [49]. Before presenting the proposed technique we recall the concepts of retrievable program, containment and maximal containment among programs.

**Definition 2.21.** Let  $\mathcal{P}$  be a program and  $\mathcal{V}$  a set of views. We say  $\mathcal{P}$  is retrievable if the only  $\mathcal{EDB}$  predicates appearing in  $\mathcal{P}$  are materialized views of  $\mathcal{V}$ .

Given two retrievable programs  $\mathcal{P}'$  and  $\mathcal{P}''$ , then  $\mathcal{P}'$  is contained in  $\mathcal{P}''$ , written  $\mathcal{P}' \sqsubseteq \mathcal{P}''$ , if for all database  $\mathcal{I}$ ,  $\mathcal{P}'(\mathcal{V}(\mathcal{I})) \subseteq_C \mathcal{P}''(\mathcal{V}(\mathcal{I}))$  where  $C$  is the set of all constants in  $\mathcal{I}$ .

Given a Datalog program  $\mathcal{P}$  and a retrievable program  $\mathcal{P}'$ , we say that  $\mathcal{P}'$  is maximally contained in  $\mathcal{P}$ , if  $\mathcal{P}' \sqsubseteq \mathcal{P}$  and there is no retrievable program  $\mathcal{P}''$  such that  $\mathcal{P}' \sqsubset \mathcal{P}'' \sqsubseteq \mathcal{P}$ .  $\square$

In [49] it was shown that given a Datalog program  $\mathcal{P}$  and a set of conjunctive views  $\mathcal{V}$  it is undecidable whether there is a retrievable program  $\mathcal{P}_{\mathcal{V}}$  equivalent to  $\mathcal{P}$ . Moreover, it was also shown that for Datalog programs and conjunctive views it is possible to generate a retrievable program  $\mathcal{P}_{\mathcal{V}}$  which is maximally contained in  $\mathcal{P}$ . The technique consists of two steps. In the first step a program that might contain function symbols is built, and in the second step the program is rewritten to eliminate function symbols.

The first step is based on the inversion of rules defining views. Given a view  $v$  of the form  $v(X) \leftarrow b_1(Y_1), \dots, b_n(Y_n)$ , the inverse of  $v$ , denoted  $v^{-1}$ , is a set of  $n$  rules of the form  $b_i(Y'_i) \leftarrow v(X)$  where  $Y'_i$  is derived from  $Y_i$  by replacing every variable  $y \in (Y_i - X)$  with the function  $f_{v/y}(X)$  where  $v$  identifies the view and  $y$  identifies a distinguished variable in the view. Given a set of views  $\mathcal{V}$ ,  $\mathcal{V}^1$  denotes the union of all inverses  $v^{-1}$  of all view definitions  $v$  in  $\mathcal{V}$ , that is  $\mathcal{V}^1 = \cup_{v \in \mathcal{V}} v^{-1}$ .

*Example 2.22.* Assume to have three base relations having the following schema  $supplier(S\#, NameS, City)$ ,  $product(P\#, NameP, Type, Price)$  and  $supply(S\#, P\#)$ . Consider the views *made* and *price* defined by the following two rules:

$$\begin{aligned} 1 : & \textit{made}(P, C) \leftarrow \textit{supplier}(S, NS, C), \textit{supply}(S, P). \\ 2 : & \textit{price}(P, Pr) \leftarrow \textit{product}(P, NP, T, Pr). \end{aligned}$$

where a tuple  $\langle p, c \rangle$  in the materialized view *made* means that the product with code  $p$  was made by a supplier of city  $c$ , whereas the materialized view *price* consists of the projection of the base relation *product* on the attribute  $P\#$  and  $Price$ . The inverse of the two views is given by the following set of rules:

$$\begin{aligned} & \textit{supplier}(f_{1/S}(P, C), f_{1/NS}(P, C), C) \leftarrow \textit{made}(P, C). \\ & \textit{supply}(f_{1/S}(P, C), P) \leftarrow \textit{made}(P, C). \\ & \textit{product}(P, f_{2/NP}(P, Pr), f_{2/T}(P, Pr), Pr) \leftarrow \textit{price}(P, Pr). \end{aligned}$$

$\square$

Now, given a Datalog program  $\mathcal{P}$  and a set of views  $\mathcal{V}$ ,  $\mathcal{P}_{\mathcal{V}^{-1}}$  denotes the program derived from the union of  $\mathcal{P}$  and  $\mathcal{V}^{-1}$ . Although  $\mathcal{P}_{\mathcal{V}^{-1}}$  contains function symbols it has a unique finite minimal model since function symbols only appear in the body of nonrecursive rules. Therefore, its fixpoint evaluation is guaranteed to terminate.

Essentially, every tuple derived from the inverted views is associated with a tuple of the materialized views which, in turn, can be derived from more than one instance of the view. Thus, a tuple derived from the inverted views has associated a set of tuples in the original definition of the predicate. Consider for instance the definition of *made* in the above example and the rules defining *supplier* and

*supply* derived from the inversion of the view *made*. Assuming that the definition of *supply* in the database consists of the two tuples  $supply(s1, p1)$  and  $supply(s2, p1)$ , and the definition of *supplier* consists of the two tuples  $\langle s1, ibm, rome \rangle$  and  $\langle s2, sun, rome \rangle$ , the materialized view *made* contains only the tuple  $\langle p1, rome \rangle$  and, from the inverted view, we derive the two tuples  $supply(f_{1/S}(p1, rome), p1)$  and  $supplier(f_{1/S}(p1, rome), f_{1/NS}(p1, rome), rome)$ .

Therefore, each tuple with function symbols has associated a set of database tuples having the same flat terms. In our example the tuple  $supplier(f_{1/S}(p1, rome), f_{1/NS}(p1, rome), rome)$  has associated two tuples in the source database, namely  $\langle s1, ibm, rome \rangle$  and  $\langle s2, sun, rome \rangle$ . This means that the body of a ground view is satisfied by using database tuples if and only if it is also satisfied by using corresponding tuples with possible function terms. The database built by using materialized views will be called rebuilt database. Moreover, for each tuple  $t$  derived from the application of a program  $\mathcal{P}$  on the rebuilt database, there is a tuple  $u$  derived from the application of  $\mathcal{P}$  to the source database coinciding with  $t$  on the flat terms of  $t$ .

The second step proposed in [49] is the elimination of function symbols by deriving an equivalent Datalog version of the rewritten program. Here we do not consider this rewriting since any rewritten program has a finite unique minimal model which coincides with its fixpoint.

In [20, 61] it is considered the problem of answering queries using materialized views in the presence of negative goals. The solution is carried out by ‘inverting’ views to derive new knowledge. In order to derive both positive and negative knowledge, are generated ‘rules’ having also, in addition to negation-as-failure, classical negation. The main difference of this framework with respect to previous works is that are considered not only positive conjunctive views, but also negation and disjunctive views; moreover functional dependencies are also analyzed in order to derive additional information.

Essentially, the technique extends the one presented in [49], allowing the existence of negated atoms in both queries and views and derive, by inverting views, both positive and negative knowledge.

The rewriting of the query is performed in two steps. In the first step, given a set of conjunctive views  $\mathcal{V}$ , we derive an extended semipositive Datalog program  $\mathcal{V}^{-1}$  extracting information about base atoms from view atoms. In the second step we rewrite the rules in the query by generating an extended positive Datalog program, that is a program whose rules contain classical negation but not negation-as-failure.

The rewriting of views produces three groups of rules:

1. The rules in the first group are used to derive positive information and they coincide with the rules described in the previous section. These rules are generated according to the idea that, if the head of the view is true then all literals in the body must also be true since we are considering conjunctive views, that is for each view predicate there is only one rule defining it. Variables appearing in

the body literals and not appearing in the head have unknown values and are replaced by functions.

2. The rules in the second group are used to derive negative information. The idea here is that a body literal must be false if the head of the view is false and all other body literals are true. Moreover, since view atoms contain only ground flat terms, for each variable  $X$  appearing in a negated view atoms, a predicate  $dom(X)$  is added in the body.
3. The rules in the third group define the predicate  $dom$ , that is the database domain.

The following example shows how our technique works.

*Example 2.23.* Consider the following view  $v$

$$v(Y) \leftarrow emp(X, S), mgr(X, Y).$$

The first group of rules, used to derive positive information, consists of one rule for each body literal:

$$\begin{aligned} emp(f_{v/X}(Y), f_{v/S}(Y)) &\leftarrow v(Y). \\ mgr(f_{v/X}(Y), Y) &\leftarrow v(Y). \end{aligned}$$

where variables not appearing in the head of views are replaced by function terms. The second group of rules is used to derive negative information. Thus, a new rule is generated for each body literal.

$$\begin{aligned} \neg emp(X, \_) &\leftarrow mgr(X, Y), not v(Y), dom(Y). \\ \neg mgr(X, Y) &\leftarrow emp(X, S), not v(Y), dom(Y). \end{aligned}$$

where the database variable  $\_$  unifies with all possible ground terms. The last group of rules are used to define the database domain:  $dom(Y) \leftarrow v(Y)$ .  $\square$

The second step of the rewriting modifies the query program by replacing each base literal  $B$  with  $ev(B)$ , where  $ev(B)$  denotes extended version of a literal  $B$ , that is  $ev(B)$  is derived from  $B$  by replacing the negation  $not$  with  $\neg$ .

Given a set of views  $\mathcal{V}$ , and a semipositive Datalog program  $\mathcal{P}$  we denote with  $\mathcal{P}_{\mathcal{V}^{-1}}^-$  the program derived from the union of  $\mathcal{P}^-$  and  $\mathcal{V}^{-1}$ , where  $\mathcal{V}^{-1}$  are the rules derived in the first step of the technique and  $\mathcal{P}^-$  the program rewritten in the second step. Moreover, for a given program  $\mathcal{P}$ ,  $\mathcal{P}_{\mathcal{V}^{-1}}^-(\mathcal{V}(D))$  denotes the application of  $\mathcal{P}_{\mathcal{V}^{-1}}^-$  to the materialized views.

Let  $\mathcal{P}$  be a semipositive Datalog program,  $\mathcal{V}$  be a set of conjunctive views with possible negation in the body and  $\mathcal{I}$  a database. Then, 1)  $\mathcal{P}_{\mathcal{V}^{-1}}^-(\mathcal{V}(D))$  has a perfect minimal model which is finite and consistent 2)  $\mathcal{P}_{\mathcal{V}^{-1}}^- \cup \mathcal{V} \sqsubseteq \mathcal{P}$ .

Although the program generated by the above technique gives a good approximation of the original program, generally it is not maximal. However, for conjunctive views with at most one negation for each rule,  $\mathcal{P}_{\mathcal{V}^{-1}}^-$  is maximally contained in  $\mathcal{P}$ . The framework is extended by considering the inversion of disjunctive views [61].

## 2.8 Minimal Change Integrity Maintenance using Tuple Deletions

In [40] a practical framework for computing consistent query answers for large, possibly inconsistent relational databases is proposed. The proposed framework handles union of conjunctive queries and can effectively (and efficiently) extract indefinite disjunctive information from an inconsistent database. The paper also describes a number of novel optimization techniques applicable in this context and summarize experimental results validating the proposal.

The problem of minimal-change integrity maintenance in the context of integrity constraints (denial constraints, general functional and inclusion dependencies, as well as key and foreign key constraints) in relational databases, has been investigated in [39]. The paper discusses the different interpretation of minimal change based on whether the information in the database is assumed to be *correct* and *complete*. If the information is complete, but not necessarily correct (it may violate integrity constraints), the only way to fix the database is to delete some part of it. If the information is both incorrect and incomplete, then both insertions and deletions should be considered. The notion of *repair* pursued in the paper reflects the assumption that the database is complete, therefore the paper assumes that integrity-restoration actions are limited to tuple deletions. This scenario is common in data warehousing applications where dirty data coming from many sources is cleaned in order to be used as a part of the warehouse itself.

Given a set of denial constraints  $F$  and an instance  $r$ , all the repairs for  $r$  can be succinctly represented as a *conflict hypergraph*,  $G_{F,f}$  that is an hypergraph whose set of vertices is the set  $\Sigma(r)$  of facts of the instance  $r$  and whose set of edges consists of all the sets

$$\{P_1(t_1) \dots P_n(t_n)\}$$

such that

$$P_1(t_1), P_2(t_2), P_3(t_3) \dots P_n(t_n) \in \Sigma(r)$$

and there is a constraint

$$\forall x_1, x_2, \dots, x_n \neg [P_1(t_1) \wedge P_2(t_2) \wedge \dots \wedge P_l(t_l) \wedge \phi(x_1, x_2, \dots, x_l)]$$

in  $F$  such that  $P_1(t_1) \wedge P_2(t_2) \wedge \dots \wedge P_l(t_l)$  violate this constraint, which means that there exists a substitution  $\varrho$  such that

$$\varrho(x_1) = t_1, \varrho(x_2) = t_2, \dots, \varrho(x_l) = t_l$$

and that  $\phi(t_1, t_2, \dots, t_n)$  is true.

The paper shows that each repair of  $r$  with respect to  $F$  corresponds to a maximal independent set in  $G_{F,f}$ .

The paper studies two basic problems: *repair checking* and *consistent query answering* within this setting. Repair checking consists in checking whether a given database

is a repair of the original database. Consistent query answer consists in providing answers that are true in every repair of the database.

The paper shows that repair checking (but not consistent query answers) are in PTIME for arbitrary FDs and acyclic IDs. The obtained results are tight in the sense that relaxing any of the above restrictions leads to co-NP hard problems. Moreover, the paper also shows that for arbitrary sets of FDs and INDs repair checking is *coNP* complete and consistent query answering is  $\Pi_2^P$ -complete.

These results shed lights on the computational feasibility of minimal-change integrity maintenance. The tractable cases should lead to practical implementations, whereas the intractability results highlight the inherent limitations of any integrity enforcement mechanism, e.g., triggers or referential constraint actions, as ways of performing minimal-change integrity maintenance using tuple deletions.





## Active Integrity Constraints

**Summary.** This chapter presents *active integrity constraints (AICs)*, an extension of integrity constraints for consistent database maintenance. An active integrity constraint is a special constraint whose body contains a conjunction of literals which must be *false* and whose head contains a disjunction of update actions representing actions (insertions and deletions of tuples) to be performed if the constraint is not satisfied (that is its body is *true*). The AICs work in a domino-like manner as the satisfaction of one AIC may trigger the violation and therefore the activation of another one. The chapter also introduces *founded repairs*, that are minimal sets of update actions that make the database consistent and are specified and “supported” by active integrity constraints. The chapter presents i) a formal declarative semantics allowing the computation of founded repairs, ii) a characterization of this semantics obtained by rewriting active integrity constraints into disjunctive logic rules, so that founded repairs can be derived from the answer sets of the derived logic program. Finally, the chapter studies the computational complexity of computing founded repairs.

### 3.1 Introduction

Integrity constraints are logical assertions on acceptable or consistent database states, and specify properties of data that need to be satisfied by valid database instances [1]. In the database world it is not unusual to have the presence of data that fail to satisfy integrity constraints. For this reason the management of inconsistent data plays a key role in all the areas in which duplicate or conflicting information is likely to occur, such as data and knowledge bases [81, 83, 101, 134, 146].

Violation of constraints, that may occur for example, during or at the end of the execution of a transaction, is classically managed by performing a “repair” of the database state that is usually limited to fixed reversal actions, such as rolling back the current operation or the entire transaction [33]. In any case, in many applications there is no way to associate the cause of the inconsistency to a specific update action. Let’s consider, for example, the violation that occurs after performing the integration of multiple independent sources; in this case the updates are typically already committed and a single update operation leading to constraint violation does not exist.

An improved approach to constraints enforcement allows to define compensating actions that correct the violation of constraints according to a well-defined semantics (*database repairs*) or to compute *consistent answers*. Informally, the computation of repairs is based on the application of minimal sets of insertions and deletions of tuples so that the resulting databases satisfy all constraints, whereas the computation of consistent answers is based on the identification of tuples satisfying integrity constraints and on the selection of tuples matching the goal.

The following example shows a situation in which inconsistencies occur.

*Example 3.1.* Consider the relation schema  $mgr(Name, Dept, Salary)$  with the functional dependency  $Dept \rightarrow Name$  which can be defined through the first order formula

$$\forall(N, N', D, S, S')[mgr(N, D, S), mgr(N', D, S') \supset N = N']$$

Consider now the inconsistent instance:  $\mathcal{I} = \{mgr(john, cs, 1000), mgr(franks, cs, 2000)\}$ . A consistent (repaired) database can be obtained by applying a minimal set of update operations; in particular it admits two repaired databases:  $\mathcal{I}_1 = \{mgr(franks, cs, 2000)\}$  obtained by applying the repair  $\mathcal{R}_1 = \{-mgr(john, cs, 1000)\}$  (that is by deleting the tuple  $mgr(john, cs, 1000)$ ) and  $\mathcal{I}_2 = \{mgr(john, cs, 1000)\}$  obtained by applying the repair  $\mathcal{R}_2 = \{-mgr(franks, cs, 2000)\}$  (that is by deleting the tuple  $mgr(franks, cs, 2000)$ ).  $\square$

The problem with such a semantics is that the repairing strategy is not defined by the database administrator, and all possible repairs are computed. Thus, in this chapter we consider a special form of integrity constraint, called *active integrity constraint* (AIC), whose body consists of a conjunction of literals which should be *false* and the head contains the actions which have to be performed if the body is *true* (that is the constraint is violated). The following example illustrates the notion of active integrity constraint.

*Example 3.2.* Consider the database of Example 3.1 and the active constraint:

$$\forall(N, N', D, S, S')[mgr(N, D, S), mgr(N', D, S'), N \neq N', S \geq S' \supset -mgr(N, D, S)].$$

Basically, it models the same functional dependency reported in the previous example, but, in addition, it states that in the case of conflicting tuples, the one with the higher salary has to be removed from the database. In this case, the constraint suggests to update the database by deleting the tuple  $mgr(franks, cs, 2000)$ . This action leads only one of the two repairs, namely  $\mathcal{R}_2$ , to be taken into account.  $\square$

Active integrity constraints are *production rules* expressed by means of first order logic formulas with a declarative semantics that allows us to compute *founded repairs*, that is minimal sets of update actions making the database consistent and

whose update actions are explicitly specified and supported. In some sense, active integrity constraints represent a restricted form of *active rules* sufficient to (declaratively) express database repairs, but without the typical problems of procedural interpretations such as the *confluence* and the *termination*.

### 3.1.1 Contribution

The contribution of this chapter consists in the formal definition of active integrity constraints and in the introduction their declarative semantics. Essentially, an active integrity constraint is an integrity constraint that specifies the update actions that can be performed when it is violated. It is composed by a conjunction of literals, called *body*, and by a disjunction of update actions, called *head*. The body represents a *condition* that should be *false*, whereas the head sets the actions that can be performed when the body is *true* (that is when the constraint is violated). An inconsistent database can be repaired by means of minimal sets of update actions, called *repairs*. The semantics here introduced allows us to identify, among all possible repairs, those whose actions are *specified* in the head of some active integrity constraint and *supported* by the database or by other updates. These repairs are called *founded repairs*. The chapter studies the properties of active integrity constraints and shows that, under the proposed semantic, each update action occurring in the head of an active integrity constraint that cannot falsify the corresponding body is useless and can be deleted. Next, it shows that the computation of founded repairs can be done by rewriting the constraints into a Datalog program and computing its stable models [66]; each stable model will represent a founded repair. As the existence of founded repairs is not guaranteed, the chapter investigates a different semantics where update actions defined by active integrity constraints are interpreted as preference conditions on the set of possible repairs (*preferable semantics*). Finally, the computational complexity is analyzed and it is shown that the complexity of computing founded repairs, preferred repairs and answers is not harder than computing *standard* repairs and answers.

### 3.1.2 Plan of the Chapter

The rest of the chapter is organized as follows. Section 3.2 recalls the formal definition of integrity constraint, repair and consistent answer and briefly reviews a general approach for the computation of repairs and consistent answers. Section 3.3 introduces active integrity constraints, presents their declarative semantics and important results about their structure. Section 3.4 shows how founded repairs can be computed by rewriting active integrity constraints into a logic program and provides results on the computational complexity of computing founded repairs and queries. This section also introduces a different interpretation of active integrity constraints (*preferable semantics*), where preferred repairs are those performing specific actions with respect to other alternative repairs, and studies the computational complexity of computing preferred founded repairs.

### 3.2 Databases and Integrity constraints

Database schemata defines the structure of data and restrictions on the form the data could have. The relationship among data are generally defined by integrity constraints such as functional dependencies and inclusion dependencies. In particular, integrity constraints are used to restrict the state a database can take and to prevent the insertion or deletion of data which could produce incorrect states.

A database  $\mathcal{I}$  has an associated schema  $\langle \mathcal{DS}, \eta \rangle$  defining its intentional properties:  $\mathcal{DS}$  defines the structure of the relations, that is their names and their attributes, and  $\eta$  contains the set of integrity constraints expressing semantic information over data. Moreover  $\mathcal{I}$  has an associated domain  $Dom$  containing the values an attribute can assume. The extension that associates to each attributes a different domain is trivial.

#### 3.2.1 Integrity Constraints

An *integrity constraint*  $r$  is a formula of the first order predicate calculus of the form:

$$(\forall X) \left[ \bigwedge_{j=1}^m b_j(X_j), \varphi(X_0) \supset \bigvee_{j=m+1}^n (\exists Z_j) b_j(X_j, Z_j) \right]$$

where, let  $X = \bigcup_{j=1}^m X_j$  and  $Z = \bigcup_{j=m+1}^n Z_j$ ,  $X_i \subseteq X$ , for  $i \in [0..n]$ , all variables in  $Z$  occur once,  $\varphi(X_0)$  is a conjunction of built-in atoms and  $b_j$ , for  $j \in [1..n]$ , are base predicates. The conjunction preceding the implication symbol is the *body* of the constraint, whereas the disjunction succeeding the implication symbol is its *head*. A database satisfies  $r$  if for each  $X$ , it makes the body *false* or the head *true*. More formally, let  $\mathcal{I}$  be a database and  $Dom$  its domain. Then  $\mathcal{I}$  satisfies  $r$ , denoted as  $\mathcal{I} \models r$ , if for each  $x \in Dom^{|X|}$ , either  $\mathcal{I} \not\models \bigwedge_{j=1}^m b_j(x_j), \varphi(x_0)$  or there exists  $z \in Dom^{|Z|}$  such that  $\mathcal{I} \models \bigvee_{j=m+1}^n b_j(x_j, z_j)$ , where  $x_i$ , for  $i \in [0..n]$ , are the corresponding instances of  $X_i$  and  $z_i$ , for  $i \in [m+1..n]$ , are the corresponding instances of  $Z_i$ . Moreover,  $\mathcal{I}$  is consistent w.r.t. a set  $\eta$  of integrity constraints, denoted as  $\mathcal{I} \models \eta$ , if it satisfies all integrity constraints in  $\eta$ . Each integrity constraint can be rewritten in the following form, obtained by moving literals from the head to the body:

$$(\forall X) \left[ \bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n (\exists Z_j) b_j(X_j, Z_j), \varphi(X_0) \supset \right]. \quad (3.1)$$

#### 3.2.2 Repairing and Querying Inconsistent Databases

In this section the formal definition of consistent database and repair is first recalled and then a mechanism for computing repairs and consistent answers for inconsistent databases is presented.

An update action is of the form  $+a(X)$  or  $-a(X)$ . The update actions  $+a(X)$  and  $-a(X)$  are *duals* of each other. We write  $\alpha^D$  to denote the update action dual to an update action  $\alpha$ . The dual operator is extended to sets of update actions as appropriate. A ground update action  $+a(t)$  states that  $a(t)$  will be inserted into the database, whereas  $-a(t)$  states that  $a(t)$  will be deleted from the database. The symbol  $\pm$  will be used as a placeholder for either  $+$  or  $-$ . Given an update action  $\alpha = +a(X)$  (resp.  $\alpha = -a(X)$ ),  $lit(\alpha)$  denotes  $a(X)$  (resp.  $not\ a(X)$ ) and  $comp(\alpha)$  denotes the literal  $not\ a(X)$  (resp.  $a(X)$ ). Clearly,  $comp(\alpha) = lit(\alpha)^D$ . The operators  $lit(\cdot)$  and  $comp(\cdot)$  are extended to sets of update actions in the standard way. We also define the inverse operators  $comp(\cdot)^{-1}$  and  $lit(\cdot)^{-1}$ . Given a set  $\mathcal{U}$  of ground update actions we define the sets  $\mathcal{U}^+ = \{a(t) \mid +a(t) \in \mathcal{U}\}$ ,  $\mathcal{U}^- = \{a(t) \mid -a(t) \in \mathcal{U}\}$ . We say that  $\mathcal{U}$  is *consistent* if it does not contain two update actions  $+a(t)$  and  $-a(t)$  (that is if  $\mathcal{U}^+ \cap \mathcal{U}^- = \emptyset$ ). Given a database  $\mathcal{I}$  and a consistent set of update actions  $\mathcal{U}$ ,  $\mathcal{I} \circ \mathcal{U}$  denotes the database  $\mathcal{I}$  updated by means of  $\mathcal{U}$ , that is  $\mathcal{I} \circ \mathcal{U} = (\mathcal{I} \cup \mathcal{U}^+) \setminus \mathcal{U}^-$ .

**Definition 3.3.** (REPAIRS) *Let  $\mathcal{I}$  be a database and  $\eta$  a set of integrity constraints. A repair for  $\langle \mathcal{I}, \eta \rangle$  is a consistent set  $\mathcal{U}$  of update actions such that*

- $\mathcal{I} \circ \mathcal{U} \models \eta$  (constraint enforcement).
- for every  $\mathcal{U}' \subseteq \mathcal{U}$  such that  $\mathcal{I} \circ \mathcal{U}' \models \eta$ ,  $\mathcal{U}' = \mathcal{U}$  (minimality of change).

The set of all repairs for  $\langle \mathcal{I}, \eta \rangle$  is denoted as  $\mathbf{R}(\mathcal{I}, \eta)$ . □

Repaired databases are consistent databases, derived from the source database by means of a minimal set of update operations. Observe that for constraints containing existentially quantified variables the set of possible repairs could be infinite in the case the domain of the database is infinite. Thus, in the rest of this section *universally quantified* or *full* integrity constraints are considered. They are of the form:

$$(\forall X) \left[ \bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n not\ b_j(X_j), \varphi(X_0) \supset \right] \quad (3.2)$$

Given a set of universally quantified constraints  $\eta$ , an integrity constraint  $r \in \eta$  and a database  $\mathcal{I}$ , a ground instance of  $r$  with respect to  $\mathcal{I}$  can be obtained by replacing variables with constants in  $Dom$  and eliminating the universal quantification. The set of all ground instances of  $r$  is denoted by  $ground(r)$ , whereas  $ground(\eta) = \bigcup_{r \in \eta} ground(r)$  denotes the set of ground instances of constraints in  $\eta$ . Clearly, for any set of universally quantified constraints  $\eta$ , the cardinality of  $ground(\eta)$  is polynomial in the size of  $Dom$ . A further restriction is that we disallow integrity constraints that admit instances with inconsistent bodies (i.e. bodies always *false*). In other words we disallow ground constraints whose body contains a literal  $a$  and a literal  $not\ a$ .

**Fact 3.4** *Let  $\mathcal{I}$  be a database,  $\eta$  a set of full integrity constraints and  $\mathcal{R}$  a repair for  $\langle \mathcal{I}, \eta \rangle$ . Then, for each  $\alpha \in \mathcal{R}$  there exists in  $ground(\eta)$  an integrity constraint of the form  $\phi \wedge comp(\alpha) \supset$  such that  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\}) \models \phi$ .*

**Proof.** Straightforward from Definition 3.3. If  $\mathcal{R}$  contains an update action  $\alpha$ , then  $\mathcal{R} \setminus \{\alpha\}$  is not a repair. Thus, there must be in  $ground(\eta)$  at least an integrity constraint of the form  $\phi \wedge comp(\alpha) \supset$  such that  $\mathcal{I} \circ \mathcal{R} \setminus \{\alpha\} \models \phi$ . This integrity constraint is satisfied by  $\mathcal{I} \circ \mathcal{R}$  and violated by  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\})$ .  $\square$

The above fact states that each update action of a repair is *necessary* to satisfy at least a ground integrity constraint.

**Definition 3.5.** Given a database  $\mathcal{I}$  and a set of integrity constraints  $\eta$ , an atom  $A$  is true (resp. false) with respect to  $\langle \mathcal{I}, \eta \rangle$  if  $A$  belongs to all repaired databases (resp. there is no repaired database containing  $A$ ). The atoms which are neither true nor false are undefined.  $\square$

Thus, true atoms occur in all repaired databases, whereas undefined atoms appear in a non empty proper subset of repaired databases. Now we can provide the definition of consistent answer to a query.

**Definition 3.6.** Given a database  $\mathcal{I}$ , a set of integrity constraints  $\eta$  and a query  $Q = (g, \mathcal{P})$ , the consistent answer of  $Q$  w.r.t.  $\langle \mathcal{I}, \eta \rangle$ , denoted as  $Q(\mathcal{I}, \eta)$ , gives three sets, denoted as  $Q(\mathcal{I}, \eta)^+$ ,  $Q(\mathcal{I}, \eta)^-$  and  $Q(\mathcal{I}, \eta)^u$ . These contain, respectively, the sets of  $g$ -facts which are true, that is belonging to  $\bigcap_{\mathcal{R} \in \mathbf{R}(\mathcal{I}, \eta)} Q(\mathcal{I} \circ \mathcal{R})$ , false, that is not belonging to  $\bigcup_{\mathcal{R} \in \mathbf{R}(\mathcal{I}, \eta)} Q(\mathcal{I} \circ \mathcal{R})$  and undefined, that is the facts which are neither true nor false.  $\square$

### 3.2.3 Repairing and Querying through Stable Models

As shown in [70], the set of repairs for a database with respect to a set of full integrity constraints can be computed by rewriting the constraints into disjunctive rules. More specifically, given a database  $\mathcal{I}$  and a set of integrity constraints  $\eta$ , the technique derives a disjunctive program  $\mathcal{DP}(\eta)$  so that the repairs for  $\mathcal{I}$  can be obtained from the stable models of  $\mathcal{DP}(\eta) \cup \mathcal{I}$ .

**Definition 3.7.** Given a full integrity constraint  $r$  of the form  $(\forall X)[\bigwedge_{j=1}^n L_j, \varphi \supset]$ , where  $L_j$  is a literal, for  $j \in [1..n]$ , and  $\varphi$  is a conjunction of built-in atoms,  $dj(r)$  denotes the expression:

$$\bigvee_{j=1}^n comp^{-1}(L_j) \leftarrow \bigwedge_{j=1}^n (L_j \vee comp^{-1}(not L_j)), \varphi$$

Given a set  $\eta$  of full integrity constraints,  $\mathcal{DP}(\eta) = \{dj(r) \mid r \in \eta\} \cup \{\leftarrow -b(X), +b(X) \mid b \text{ is a predicate symbol}\}$ .  $\square$

The expression presented in the above definition is used as shorthand for a set of disjunctive rules.

Given an interpretation  $\mathcal{M}$ ,  $UpdateAtoms(\mathcal{M})$  denotes the set of update actions in  $\mathcal{M}$ . The definition of this operator is extended to sets of interpretations.

The following theorem, showing that the technique is correct and complete, has been proved in [70].

**Theorem 3.8.** *Given a database  $\mathcal{I}$  and a set  $\eta$  of full integrity constraints,*

$$\mathbf{R}(\mathcal{I}, \eta) = UpdateAtoms(\mathcal{SM}(\mathcal{DP}(\eta) \cup \mathcal{I})) \quad \square$$

The previous theorem states that for each database  $\mathcal{I}$  and set of full integrity constraints  $\eta$ :

- for every stable model  $\mathcal{M}$  of  $\mathcal{DP}(\eta) \cup \mathcal{I}$ ,  $UpdateAtoms(\mathcal{M})$  is a repair for  $\langle \mathcal{I}, \eta \rangle$  (*soundness*);
- for every repair  $\mathcal{R}$  for  $\langle \mathcal{I}, \eta \rangle$  there exists a stable model  $\mathcal{M}$  for  $\mathcal{DP}(\eta) \cup \mathcal{I}$  such that  $\mathcal{R} = UpdateAtoms(\mathcal{M})$  (*completeness*).

This technique can be used to compute consistent answers to queries. Given a database  $\mathcal{I}$ , a set of integrity constraints  $\eta$  and a query  $Q = (g, \mathcal{P})$ , the consistent answer of  $Q$  w.r.t.  $\langle \mathcal{I}, \eta \rangle$  can be computed by considering the stable models of  $\mathcal{MP}(g, \mathcal{P}) \cup \mathcal{DP}(\eta) \cup \mathcal{I}$ , where  $\mathcal{MP}(g, \mathcal{P})$  is obtained from  $\mathcal{P}$  by replacing every base predicate symbol  $p$  with  $p'$  and by adding a rule of the form  $p'(X) \leftarrow (p(X) \wedge not -p(X)) \vee +p(X)$ . Moreover, the rule  $g'(X) \leftarrow (g(X) \wedge not -g(X)) \vee +g(X)$  is added. We have that

- $Q(\mathcal{I}, \eta)^+ = \bigcap_{\mathcal{M} \in \mathcal{SM}(\mathcal{MP}(g, \mathcal{P}) \cup \mathcal{DP}(\eta) \cup \mathcal{I})} \mathcal{M}(g')$ ,
- $Q(\mathcal{I}, \eta)^u = \bigcup_{\mathcal{M} \in \mathcal{SM}(\mathcal{MP}(g, \mathcal{P}) \cup \mathcal{DP}(\eta) \cup \mathcal{I})} \mathcal{M}(g') - Q(\mathcal{I}, \eta)^+$ .

### 3.3 Active Integrity Constraints

In this section we present *active integrity constraints*, an extension of integrity constraints that allows a specification of the actions to be performed to make the database consistent.

**Definition 3.9.** *A (full) Active Integrity Constraint (AIC) is of the form*

$$(\forall X) \left[ \bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n not b_j(X_j), \varphi(X_0) \supset \bigvee_{i=1}^p \pm a_i(Y_i) \right] \quad (3.3)$$

where, let  $X = \bigcup_{j=1}^m X_j$ ,  $X_i \subseteq X$ , for  $i \in [0..n]$ , and  $Y_i \subseteq X$ , for  $i \in [1..p]$ .

□

Given an active integrity constraint  $r$  of the form 4.1 we denote the set  $\{b_1(X_1), \dots, b_m(X_m), \text{not } b_{m+1}(X_{m+1}), \dots, \text{not } b_n(X_n)\}$  as  $\text{body}(r)$  and the set  $\{\pm a(Y_1), \dots, \pm a(Y_p)\}$  as  $\text{head}(r)$ . As in the case of integrity constraints, we disallow AICs that admit instances with inconsistent bodies.

*Example 3.10.* The active integrity constraint of Example 3.2 states that in the case of conflicting tuples (there are two different managers managing the same department), we prefer to repair the database by deleting the one with the higher salary, whereas the constraint  $\forall(N, N', D, S, S')[\text{mgr}(N, D, S), \text{mgr}(N', D, S'), N \neq N' \supset -\text{mgr}(N, D, S') \vee -\text{mgr}(N', D, S')]$  states that between two different managers of the same department we do not have any preference and, therefore, one of them, selected nondeterministically, can be deleted.  $\square$

An active integrity constraint is an integrity constraint that specifies the update actions that can be performed when it is violated. The conjunction of literals in its body represents a *condition* that should be *false*, whereas update actions in its head represents the possible updates that can be performed when the constraint is violated. Given an active integrity constraint  $r$  of the form (4.1),  $ic(r)$  denotes the corresponding integrity constraint of the form (3.2) obtained from  $r$  by removing the disjunction of update action. The definition of this operator is extended to sets of active integrity constraints. A database  $\mathcal{I}$  satisfies an active integrity constraint  $r$  ( $\mathcal{I} \models r$ ) if it satisfies the corresponding integrity constraint  $ic(r)$  ( $\mathcal{I} \models ic(r)$ ). The operator  $\text{ground}(\cdot)$  for active integrity constraints is defined in the standard way.

**Definition 3.11.** *Given a database  $\mathcal{I}$  and a set of active integrity constraints  $\eta$ , a repair for  $\langle \mathcal{I}, \eta \rangle$  is any repair for  $\langle \mathcal{I}, ic(\eta) \rangle$ . The set of all repairs for  $\langle \mathcal{I}, \eta \rangle$  is denoted by  $\mathbf{R}(\mathcal{I}, \eta)$ .*  $\square$

From the previous definition, for each database  $\mathcal{I}$  and set  $\eta$  of active integrity constraints,  $\mathbf{R}(\mathcal{I}, \eta) = \mathbf{R}(\mathcal{I}, ic(\eta))$ .

Not all repairs contain atoms which can be derived from the active integrity constraints. Thus, we identify a class of repairs, called *founded*, whose actions can be derived from the active integrity constraints.

*Example 3.12.* Consider the database  $\mathcal{I} = \{\text{movie}(\text{Marshall}, \text{Chicago}, 2002), \text{director}(\text{Stone})\}$  and the active integrity constraint

$$\forall(D, T, A) [\text{movie}(D, T, A), \text{not } \text{director}(D) \supset +\text{director}(D)]$$

There are two repairs  $\mathcal{R}_1 = \{-\text{movie}(\text{Marshall}, \text{Chicago}, 2002)\}$  and  $\mathcal{R}_2 = \{+\text{director}(\text{Marshall})\}$ , but only  $\mathcal{R}_2$  contains updates “supported” by the active integrity constraint.  $\square$

**Definition 3.13.** (FOUNDED REPAIR) *Let  $\mathcal{I}$  be a database,  $\eta$  a set of active integrity constraints and  $\mathcal{R}$  a repair for  $\langle \mathcal{I}, \eta \rangle$ .*



- An update action  $\alpha \in \mathcal{R}$  is founded with respect to  $\langle \mathcal{I}, \eta \rangle$  and  $\mathcal{R}$  if there exists  $r \in \text{ground}(\eta)$  such that  $\alpha \in \text{head}(r)$  and  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\}) \models \text{body}(r)$ .  
We say that  $r$  supports  $\alpha$ .
- $\mathcal{R}$  is founded with respect to  $\langle \mathcal{I}, \eta \rangle$  if all its update actions are founded with respect to  $\langle \mathcal{I}, \eta \rangle$  and  $\mathcal{R}$ .

The set of founded repairs for  $\langle \mathcal{I}, \eta \rangle$  is denoted by  $\mathbf{FR}(\mathcal{I}, \eta)$ . □

In the previous definition, the update action  $\alpha$  is founded if two conditions are verified: i) it belongs to the head of an active integrity constraint  $r$  and; ii) if we discard it, the database updated by means of the remaining update actions ( $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\})$ ), violates  $r$ . This means that  $\alpha$  is inferred by  $r$  (because it belongs to its head) and it is *necessary* to repair the database in order to satisfy  $r$  (because if we discard it, the database violates  $r$ ).

Clearly, the set of founded repairs is contained in the set of repairs that is  $\mathbf{FR}(\mathcal{I}, \eta) \subseteq \mathbf{R}(\mathcal{I}, \eta)$ .

We introduce some additional notation useful in the following. The set of founded update actions in  $\mathcal{R}$  with respect to  $\langle \mathcal{I}, \eta \rangle$  is denoted as  $\text{Founded}(\mathcal{R}, \mathcal{I}, \eta)$ , whereas  $\text{Unfounded}(\mathcal{R}, \mathcal{I}, \eta) = \mathcal{R} \setminus \text{Founded}(\mathcal{R}, \mathcal{I}, \eta)$ . The set of active integrity constraints in  $\text{ground}(\eta)$  supporting update actions in  $\mathcal{R}$  is denoted as  $\text{Applied}(\mathcal{R}, \mathcal{I}, \eta)$ , whereas  $\text{Unapplied}(\mathcal{R}, \mathcal{I}, \eta) = \text{ground}(\eta) \setminus \text{Applied}(\mathcal{R}, \mathcal{I}, \eta)$ .

*Example 3.14.* Consider the following set  $\eta$  of active integrity constraints:

$$\begin{aligned} & \forall (E, P, D) [ mgr(E, P), prj(P, D), not emp(E, D) \supset +emp(E, D) ], \\ & \forall (E, D, D') [ emp(E, D), emp(E, D'), D \neq D' \supset -emp(E, D) \vee -emp(E, D') ] \end{aligned}$$

The first constraint states that every manager  $E$  of a project  $P$ , carried out by a department  $D$ , must be an employee of  $D$ , whereas the second one says that every employee must be in only one department. Consider now the database  $\mathcal{I} = \{mgr(e_1, p_1), prj(p_1, d_1), emp(e_1, d_2)\}$ . There are three repairs for  $\mathcal{I}$ :  $\mathcal{R}_1 = \{-mgr(e_1, p_1)\}$ ,  $\mathcal{R}_2 = \{-prj(p_1, d_1)\}$  and  $\mathcal{R}_3 = \{+emp(e_1, d_1), -emp(e_1, d_2)\}$ .  $\mathcal{R}_3$  is the only founded repair as only the update atoms  $+emp(e_1, d_1)$  and  $-emp(e_1, d_2)$  are derivable from  $\eta$ . □

**Proposition 3.15.** *Let  $\mathcal{I}$  be a database,  $\eta$  a set of active integrity constraints and  $\mathcal{R}$  a founded repair for  $\langle \mathcal{I}, \eta \rangle$ . For each ground active integrity constraint  $r = \phi \supset \psi \in \text{Applied}(\mathcal{R}, \mathcal{I}, \eta)$ ,  $\mathcal{I} \circ \text{head}(r) \not\models \phi$  (that is  $\mathcal{I} \circ \text{head}(r) \models r$ ).*

**Proof.** Let  $r = \phi \supset \psi$  be a ground AIC in  $\text{Applied}(\mathcal{R}, \mathcal{I}, \eta)$ . By definition,  $\psi$  is in the form of  $\psi' \vee \alpha$ , with  $\psi'$  a (possibly empty) disjunction of ground update actions and  $\alpha$  a ground update action supported by  $r$ .

As  $r$  is applied, we have that  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\}) \models \phi$ . Moreover, since  $\mathcal{I} \circ \mathcal{R} \not\models \phi$  we have that  $\phi$  must be of the form of  $\phi' \wedge \text{comp}(\alpha)$ , with  $\phi'$  a (possibly empty) conjunction of ground literals.

As  $\alpha \in \text{head}(r)$  and  $\phi = \phi' \wedge \text{comp}(\alpha)$ , it follows that  $\mathcal{I} \circ \text{head}(r) \not\models \phi$ .  $\square$

The above proposition states that for each ground applied constraint there must be among the true update head atoms, at least one atom  $\alpha$  which is used to repair the database with respect to the body of the rule, that is the body must contain a literal  $\text{comp}(\text{head}(r))$ . Observe that, if for each ground AIC  $r$ ,  $\text{head}(r)$  is such that  $\mathcal{I} \circ \text{head}(r) \models \phi$  (that is  $\mathcal{I} \circ \text{head}(r) \not\models r$ ), no founded repair exists.

Now we start our analysis of the structure of active integrity constraints by introducing the concept of *Core*.

**Definition 3.16.** Given a ground AIC  $r = \phi \supset \psi$ ,  $\text{Core}(r)$  denotes the ground AIC  $\phi \supset \psi'$ , where  $\psi'$  is obtained by deleting from  $\psi$  any update action  $\alpha$  such that  $\text{comp}(\alpha) \notin \text{body}(r)$ .  $\square$

The definition of  $\text{Core}(\cdot)$  is extended to sets of ground AICs. Moreover, given a non-ground set  $\eta$  of AICs,  $\text{Core}(\eta) = \text{Core}(\text{ground}(\eta))$ . The following theorem shows the equivalence between a set of active integrity constraints and its *Core*.

**Theorem 3.17.** Given a database  $\mathcal{I}$  and a set  $\eta$  of active integrity constraints,

$$\mathbf{FR}(\mathcal{I}, \eta) = \mathbf{FR}(\mathcal{I}, \text{Core}(\eta)).$$

**Proof.** As  $\mathbf{FR}(\mathcal{I}, \eta) = \mathbf{FR}(\mathcal{I}, \text{ground}(\eta))$  we prove that  $\mathbf{FR}(\mathcal{I}, \text{ground}(\eta)) = \mathbf{FR}(\mathcal{I}, \text{Core}(\eta))$ .

1. Firstly we prove that  $\mathbf{FR}(\mathcal{I}, \text{ground}(\eta)) \subseteq \mathbf{FR}(\mathcal{I}, \text{Core}(\eta))$ .

Let  $r = \phi \supset \psi$  be a constraint in  $\text{ground}(\eta)$ . Let  $\psi'$  be the disjunction of update actions  $\alpha$  occurring in  $\psi$  such that the literal  $\text{comp}(\alpha)$  occurs in  $\phi$  and  $\psi''$  the disjunction of remaining update actions appearing in  $\psi$ . Then  $r$  is of the form  $\phi \supset \psi' \vee \psi''$ . Let  $r' = \phi \supset \psi'$  and  $\eta' = (\text{ground}(\eta) \setminus \{r\}) \cup \{r'\}$ . We prove that  $\mathbf{FR}(\mathcal{I}, \text{ground}(\eta)) \subseteq \mathbf{FR}(\mathcal{I}, \eta')$ , that is that for each  $\mathcal{R} \in \mathbf{FR}(\mathcal{I}, \text{ground}(\eta))$ , also  $\mathcal{R} \in \mathbf{FR}(\mathcal{I}, \eta')$  holds. As  $\text{ic}(\text{ground}(\eta)) = \text{ic}(\eta')$ , and so  $\mathbf{R}(\mathcal{I}, \text{ground}(\eta)) = \mathbf{R}(\mathcal{I}, \eta')$ , it follows that  $\mathcal{R} \in \mathbf{R}(\mathcal{I}, \eta')$ . Therefore, we have just to prove that  $\mathcal{R}$  is founded with respect to  $\langle \mathcal{I}, \eta' \rangle$ . For each  $\alpha \in \mathcal{R}$ , there exists in  $\text{ground}(\eta)$  an AIC  $g$  supporting  $\alpha$ , that is such that  $\alpha \in \text{head}(g)$  and  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\}) \models \text{body}(g)$ . There are two cases: either  $g \neq r$  or  $g = r$ . If  $g \neq r$  then  $g \in \eta'$  and it supports  $\alpha$ . If  $g = r$  then  $g \notin \eta'$ , but  $r' = \phi \supset \psi' \in \eta'$ . As  $\mathcal{I} \circ \mathcal{R} \not\models \phi$  and  $\mathcal{I} \circ \mathcal{R}' \models \phi$ , it follows that  $\phi = \phi' \wedge \text{comp}(\alpha)$  where  $\phi'$  is a (possibly empty) conjunction of ground literals such that  $\mathcal{I} \circ \mathcal{R} \models \phi'$ . Thus,  $\alpha$  occurs in  $\psi'$  and it is supported by  $r'$ . This step can be repeated to obtain  $\text{Core}(\eta)$  from  $\text{ground}(\eta)$ .

2. Now we prove that  $\mathbf{FR}(\mathcal{I}, \text{Core}(\eta)) \subseteq \mathbf{FR}(\mathcal{I}, \text{ground}(\eta))$ .

Let  $\mathcal{R} \in \mathbf{FR}(\mathcal{I}, \text{Core}(\eta))$ . We will prove that  $\mathcal{R} \in \mathbf{FR}(\mathcal{I}, \text{ground}(\eta))$ . As  $\mathcal{R}$  is a repair for  $\langle \mathcal{I}, \text{ground}(\eta) \rangle$ , it is sufficient to prove that  $\mathcal{R}$  is founded. Let  $r = \phi \supset \psi$  be a constraint in  $\text{Core}(\eta)$  and  $r' = \phi \supset \psi \vee \alpha$ , where  $\alpha$  is an

update action. Let  $\eta' = (\text{ground}(\eta) \setminus \{r\}) \cup \{r'\}$ . Obviously, each update action in  $\mathcal{R}$  is again founded. Thus,  $\mathcal{R}$  is a founded repair for  $\langle \mathcal{I}, \eta' \rangle$ . This step can be repeated to obtain  $\text{ground}(\eta)$  from  $\text{Core}(\eta)$ .  $\square$

The above Theorem state that every update action  $\alpha$  occurring in the the head of an AIC  $r$  that cannot repair directly the body of  $r$ , that is such that the body does not contain a literal  $\text{comp}(\alpha)$ , is useless and can be deleted. This is an important result as it shows that production rules with the declarative semantics proposed here, should have a specific form: the head update actions must repair databases so that the body of the corresponding active constraints is *true*.

*Example 3.18.* Consider the database  $\mathcal{I} = \{a, b\}$  and the set  $\eta = \{a \supset -b, b \supset -a\}$  of active integrity constraints. The unique repair for  $\langle \mathcal{I}, \eta \rangle$  is  $\mathcal{R} = \{-a, -b\}$ , but it is not founded. Intuitively, if we apply  $a \supset -b$ ,  $b$  is deleted from  $\mathcal{I}$ , so  $b \supset -a$  cannot be applied. If we apply  $b \supset -a$ ,  $a$  is deleted from  $\mathcal{I}$ , so  $a \supset -b$  cannot be applied.  $\square$

Thus, in the following, only ground AICs where for each head update action  $\alpha$ , there exists in the body a corresponding complementary literal  $\text{comp}(\alpha)$ , are considered.

**Theorem 3.19.** *Let  $\mathcal{I}$  be a database,  $\eta$  a set of active integrity constraints and  $\mathcal{R}$  a founded repair for  $\langle \mathcal{I}, \eta \rangle$ . Then for each  $\alpha \in \mathcal{R}$ , there exists an active integrity constraint  $r \in \text{Core}(\eta)$  such that  $\alpha$  is the unique update action in  $\mathcal{R}$  supported by  $r$ .*  $\square$

**Proof.** (by contradiction)

Let  $\alpha \in \mathcal{R}$ . As  $\alpha$  is founded, there exists at least one AIC in  $\text{Core}(\eta)$  supporting  $\alpha$ . Let  $\mathcal{G}$  be the set of AICs in  $\text{Core}(\eta)$  supporting  $\alpha$  and let us suppose, by contradiction, that each of these AICs supports at least two update actions in  $\mathcal{R}$ . The AICs in  $\mathcal{G}$  are of the form

$$\begin{aligned} g_1 &: \text{comp}(\alpha), \text{comp}(\alpha_1), \phi_1 \supset \alpha \vee \alpha_1 \vee \psi_1 \\ &\dots \\ g_n &: \text{comp}(\alpha), \text{comp}(\alpha_n), \phi_n \supset \alpha \vee \alpha_n \vee \psi_n \end{aligned}$$

where, for  $i \in [1..n]$ ,  $\psi_i$  is a disjunction of ground update actions,  $\phi_i$  is a conjunction of ground literals and  $\alpha_i$  is a ground update action in  $\mathcal{R}$  supported by  $g_i$ . We observe that, for  $i \in [1..n]$ ,  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha_i\}) \not\models \text{body}(g_i)$  as  $\alpha \in \mathcal{R}$  and  $\text{comp}(\alpha) \in \text{body}(g_i)$ , thus  $\alpha_i$  is not supported by  $g_i$ .  $\square$

### Normalization

**Definition 3.20.** *Given an active integrity constraint  $r$  of the form (4.1),  $\text{Normalized}(r)$  denotes the set of active integrity constraints*

$$(\forall X) \left[ \bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n \text{not } b_j(X_j), \varphi(X_0) \supset \pm a_i(Y_i) \right]$$

for  $i \in [1..p]$ . The operator  $Normalized(\cdot)$  is extended to sets of active integrity constraints in the standard way.  $\square$

Given a database  $\mathcal{I}$  and a set of active integrity constraints  $\eta$ , the set of repairs for  $\langle \mathcal{I}, \eta \rangle$  coincides with the set of repairs for  $\langle \mathcal{I}, Normalized(\eta) \rangle$  (i.e.  $\mathbf{R}(\mathcal{I}, \eta) = \mathbf{R}(\mathcal{I}, Normalized(\eta))$ ) as  $ic(\mathcal{I}) = ic(Normalized(\mathcal{I}))$ . This property is pretty obvious; however it can be extended to the case of founded repairs.

**Proposition 3.21.** *Given a database  $\mathcal{I}$  and a set  $\eta$  of active integrity constraints,*

$$\mathbf{FR}(\mathcal{I}, \eta) = \mathbf{FR}(\mathcal{I}, Normalized(\eta)).$$

**Proof.**

1. Firstly we prove that  $\mathbf{FR}(\mathcal{I}, \eta) \subseteq \mathbf{FR}(\mathcal{DB}, Normalized(\eta))$ .  
Let  $\mathcal{R} \in \mathbf{FR}(\mathcal{I}, \eta)$ . Obviously,  $\mathcal{R} \in \mathbf{R}(\mathcal{DB}, Normalized(\eta))$ . Thus, we have to prove that it is founded with respect to  $\langle \mathcal{I}, Normalized(\eta) \rangle$ . By definition, for each  $\pm a(t) \in \mathcal{R}$ , there exists in  $ground(\eta)$  a ground active integrity constraint  $\phi \wedge comp(\alpha) \supset \psi \vee \alpha$  such that  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\}) \models \phi$ . Thus,  $ground(Normalized(\eta))$  contains the ground active integrity constraint  $\phi \wedge comp(\alpha) \supset \alpha$  which supports  $\alpha$ .
2. Now we prove that  $\mathbf{FR}(\mathcal{I}, Normalized(\eta)) \subseteq \mathbf{FR}(\mathcal{I}, \eta)$ .  
Let  $\mathcal{R} \in \mathbf{FR}(\mathcal{I}, Normalized(\eta))$ . Obviously,  $\mathcal{R} \in \mathbf{R}(\mathcal{I}, \eta)$ . Thus, we have to prove that it is founded with respect to  $\langle \mathcal{I}, \eta \rangle$ . By definition, for each  $\alpha \in \mathcal{R}$  there exists in  $ground(Normalized(\eta))$  a ground active integrity constraint  $\phi \wedge comp(\alpha) \supset \alpha$  such that  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\}) \models \phi$ . Thus,  $ground(\eta)$  contains a ground active integrity constraint  $\phi \wedge comp(\alpha) \supset \psi \vee \alpha$  which supports  $\alpha$ .  $\square$

This theorem states that each active integrity constraint having  $p$  update atoms in the head, can be *unpacked* into  $p$  active integrity constraints having a single update atom in the head. Therefore, there is no loss of generality in considering active integrity constraints having just one update atom in the head.

### Conditioned Active Integrity Constraints

As in the ground set of active integrity constraints we only consider actions which make the body *false*, every non ground active integrity constraint can be seen as an active constraint where actions have associated a condition defining its applicability.

**Definition 3.22.** *A (full) Conditioned Active Integrity Constraint (CAIC) is of the form*

$$(\forall X) \left[ \bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n \text{not } b_j(X_j), \varphi(X_0) \supset \bigvee_{i=1}^p \varphi_i(Z_i) \wedge \pm a_i(Y_i) \right] \quad (3.4)$$

where, let  $X = \bigcup_{j=1}^m X_j$ ,  $X_i \subseteq X$  for  $i \in [0..n]$ ,  $Y_i, Z_i \subseteq X$  for  $i \in [1..p]$  and  $\varphi_i(Z_i)$ , for  $i \in [0..p]$ , is a first order formula of built-in atoms.  $\square$

Every conjunction  $\varphi_i(Z_i) \wedge \pm a_i(Y_i)$  appearing in the head of the constraint is called *conditioned update atom* and its intuitive meaning is that the action  $\pm a_i(Y_i)$  can be performed only if the condition  $\varphi_i(Z_i)$  is *true*.

**Definition 3.23.** Given an AIC

$$r = (\forall X) \left[ \Phi \supset \bigvee_{i=1}^p \pm a_i(Y_i) \right],$$

*Conditioned(r)* denotes the CAIC

$$(\forall X) \left[ \Phi \supset \bigvee_{i=1}^p (\bigvee_{\text{comp}(\pm a_i(Z)) \in \text{body}(r)} Y_i = Z) \wedge \pm a_i(Y_i) \right]$$

The operator *Conditioned(·)* is extended to sets of active integrity constraints in the standard way.  $\square$

Thus, *Conditioned(r)* replaces every update action  $\pm a_i(Y_i)$  with a conditioned update action  $((Z_1 = Y_i) \vee \dots \vee (Z_k = Y_i)) \wedge \pm a_i(Y_i)$  so that the update action can be applied only if the body of  $r$  contains a literal  $\text{comp}(\pm a_i(Z_h))$  and  $Z_h = Y_i$ , for  $h \in [1..k]$ .

*Example 3.24.* Given the AIC  $r = p(a, X), q(Y) \supset -p(Y, b)$ ,

$$\text{Conditioned}(r) = p(a, X), q(Y) \supset ((Y, b) = (a, X)) \wedge -p(Y, b) \quad \square$$

We observe that a ground instance of a CAIC is a ground AIC as the evaluation of the conditions in the head does not depend on the database instance. Therefore, for any ground CAIC  $r$ , every (ground) conditioned head update action  $\varphi \wedge \alpha$  can be deleted if  $\varphi$  is *false* or replaced by  $\alpha$  if  $\varphi$  is *true*.

**Proposition 3.25.** Given a set  $\eta$  of active integrity constraints, for each database  $\mathcal{I}$

$$\text{ground}(\text{Conditioned}(\eta)) = \text{Core}(\eta)$$

**Proof.**

- Firstly, we prove that  $\text{ground}(\text{Conditioned}(\eta)) \subseteq \text{Core}(\eta)$ . Let  $s = \Phi \supset \bigvee_{i=1}^p \alpha_i$  a ground AIC belonging to  $\text{ground}(\text{Conditioned}(\eta))$  and  $r \in \eta$  such that  $s \in \text{ground}(\text{Conditioned}(r))$ . From Definition 3.23 it follows that for each  $i \in [1..p]$ , the literal  $\text{comp}(\alpha_i)$  occurs in  $\Phi$ . Therefore, from Definition 3.16,  $s \in \text{Core}(r)$ .

- Now, we prove that  $Core(\eta) \subseteq ground(Conditioned(\eta))$ .  
Let  $s = \Phi \supset \bigvee_{j=1}^p \alpha_j$  a ground AIC belonging to  $Core(\eta)$  and  $r \in \eta$  such that  $s \in Core(r)$ . From Definition 3.16 it follows that for each  $i \in [1..p]$ , the literal  $comp(\alpha_i)$  occurs in  $\Phi$ . Therefore, from Definition 3.23,  $s \in Conditioned(r)$ .  $\square$

Essentially, the previous proposition shows a direct way of *pushing* the syntactic restriction of the Core into nonground AICs.

*Example 3.26.* Given the active integrity constraint

$$r = p(X, a), q(Y) \supset -p(X, Y)$$

and the database  $\mathcal{I} = \{q(a), q(b)\}$ ,  $Conditioned(r)$  is

$$p(X, a), q(Y) \supset ((X, Y) = (X, a)) \wedge -p(X, Y).$$

The AICs in  $ground(Conditioned(r))$  are

$$\begin{aligned} p(a, a), q(a) &\supset -p(a, a) \\ p(a, a), q(b) &\supset \\ p(b, a), q(b) &\supset \\ p(b, a), q(a) &\supset -p(b, a) \end{aligned}$$

This set of active integrity constraints coincides with  $Core(r)$ .  $\square$

Given a set  $\eta$  of conditioned active integrity constraints and a database  $\mathcal{I}$ , we define the founded repairs for  $\langle \mathcal{I}, \eta \rangle$  as the founded repairs for  $\langle \mathcal{I}, ground(\eta) \rangle$ . Therefore, the following fact holds.

**Fact 3.27** *Given a set  $\eta$  of active integrity constraints and a database  $\mathcal{I}$*

$$FR(\mathcal{I}, \eta) = FR(\mathcal{I}, Conditioned(\eta)).$$

### 3.4 Computation and Complexity

As shown before, a general approach for the computation of repairs for a database  $\mathcal{I}$  with respect to a set of full integrity constraints  $\eta$  has been proposed in [71]. The technique is based on the generation of a disjunctive program  $\mathcal{DP}(\eta)$  derived from  $\eta$  so that the repairs can be derived from the stable models of  $\mathcal{DP}(\eta) \cup \mathcal{I}$ .

Such a technique cannot be easily adapted to active integrity constraints by simply putting in the head of  $d_j^i(r)$  only the atoms appearing in the head of the corresponding active constraint  $r$ . To intuitively show this, consider the database  $\mathcal{I} = \{a, b\}$  and the set of active integrity constraints  $\eta = \{a \supset -a, a, b \supset -b\}$ . The database  $\mathcal{I}$  is inconsistent and the unique founded repair is  $\mathcal{R} = \{-a\}$ . Moreover, considering the rewriting function  $\mathcal{DP}'$  which puts in the head of logic rules only the update actions appearing in the head of integrity constraints, we have that the program  $\mathcal{DP}'(\eta)$  consists of the rules  $-a \leftarrow (a \vee +a)$  and  $-b \leftarrow (a \vee +a), (b \vee +b)$ . The program  $\mathcal{DP}'(\eta) \cup \mathcal{I}$  has a unique stable model  $\mathcal{M} = \{-a, -b, a, b\}$  from which we derive the set of updates  $UpdateAtoms(\mathcal{M}) = \{-a, -b\}$  which is not a repair.

### 3.4.1 Rewriting into Logic Programs

A different technique, which generalizes the one proposed in [70, 71], so that founded repairs can be computed by logic programs derived from active integrity constraints, will be now presented. It is worth noting that the presence of existentially quantified variables in negated body literals, does not allow the generation of a possibly infinite number of repairs as the logic rules derived from the rewriting of constraints are *safe* [139]. Moreover, for the sake of simplicity we only consider universally quantified constraints.

**Definition 3.28.** Let  $c$  be a (range restricted) AIC of the form

$$(\forall X)[ \bigwedge_{j=1}^n L_j, \varphi \supset \bigvee_{i=1}^p \alpha_i ]$$

we denote as  $fp(c)$  the set of constraints

$$\leftarrow comp^{-1}(L_j), not\ comp^{-1}(L_j^f) \quad j \in [1..n]$$

and of rules

$$\alpha_i^f \leftarrow \bigwedge_{j=1}^n (L_j \wedge (not\ comp^{-1}(L_j) \vee \alpha_i = comp^{-1}(L_j)) \vee comp^{-1}(not\ L_j)), \varphi \quad i \in [1..p]$$

where let  $comp^{-1}(L_j) = \pm a(X)$ , for  $j \in [1..n]$ ,  $comp^{-1}(L_j^f) = \pm a^f(X)$ . Given a set of active integrity constraints  $\eta$ , we define  $\mathcal{FP}(\eta) = \bigcup_{c \in \eta} fp(c)$ , and  $\mathcal{FDP}(\eta) = \mathcal{DP}(ic(\eta)) \cup \mathcal{FP}(\eta)$ .  $\square$

Observe that in the above definition, let  $\alpha_i$  and  $comp^{-1}(L_j)$  be equal, respectively, to  $\pm a_i(X_i)$  and  $\pm b_j(Y_j)$ , the equality  $\pm a_i(X_i) = \pm b_j(Y_j)$  is just shorthand for  $X_i = Y_j$  if  $\pm a_i = \pm b_j$  and *false* otherwise.

*Example 3.29.* Given the database  $\mathcal{I} = \{p(a), p(b), q(a)\}$  and the AIC

$$r = p(a), p(b), q(X) \supset -p(X)$$

Then  $\mathcal{DP}(St(r))$  consists of the rules

$$\begin{aligned} -p(a) \vee -p(b) \vee -q(X) &\leftarrow (p(a) \vee +p(a)), (p(b) \vee +p(b)), (q(X) \vee +q(X)) \\ &\leftarrow -p(X) \wedge +p(X) \\ &\leftarrow -q(X) \wedge +q(X) \end{aligned}$$

whereas, the set  $\mathcal{FP}(r)$  is equal to the set of rules

$$\begin{aligned} &\leftarrow -p(a), not\ -p^f(a); \\ &\leftarrow -p(b), not\ -p^f(b); \\ &\leftarrow -q(X), not\ -q^f(X); \end{aligned}$$

$$\begin{aligned}
-p^f(X) \leftarrow & ((p(a) \wedge (\text{not } -p(a) \vee X = a)) \vee +p(a)), \\
& ((p(b) \wedge (\text{not } -p(b) \vee X = b)) \vee +p(b)), \\
& ((q(X) \wedge \text{not } -q(X)) \vee +q(X))
\end{aligned}$$

□

Next theorem shows that the rewriting technique is sound and complete.

**Theorem 3.30.** *Given a database  $\mathcal{I}$  and a set  $\eta$  of active integrity constraints,*

$$\mathbf{FR}(\mathcal{I}, \eta) = \text{UpdateAtoms}(\mathcal{SM}(\mathcal{FDP}(\eta) \cup \mathcal{I}))$$

**Proof.**

- (Soundness) Firstly, we prove that  $\text{UpdateAtoms}(\mathcal{SM}(\mathcal{FDP}(\eta) \cup \mathcal{I})) \subseteq \mathbf{FR}(\mathcal{I}, \eta)$ , that is that for each  $\mathcal{M} \in \mathcal{SM}(\mathcal{FDP}(\eta) \cup \mathcal{I})$  there exists a repair  $\mathcal{R} \in \mathbf{FR}(\mathcal{I}, \eta)$  such that  $\text{UpdateAtoms}(\mathcal{M}) = \mathcal{R}$ .

Let  $\mathcal{M}$  be a stable model of  $\mathcal{FDP}(\eta) \cup \mathcal{I}$ ,  $\mathcal{M}^r = \text{UpdateAtoms}(\mathcal{M}) \cup \mathcal{I}$  and  $\mathcal{M}^f = \mathcal{M} \setminus \mathcal{M}^r$  (the set of atoms defined by  $\mathcal{FP}(\eta)$ ).  $\mathcal{M}^r$  is a stable model of  $\mathcal{DP}(\text{ic}(\eta)) \cup \mathcal{I}$ . This holds because update actions can be inferred just from rules in  $\mathcal{DP}(\text{ic}(\eta)) \cup \mathcal{I}$  and the body of these rules do not contain primed atoms. As  $\text{UpdateAtoms}(\mathcal{M}^r)$  is a repair for  $\langle \mathcal{I}, \text{ic}(\eta) \rangle$  (Theorem 3.8),  $\text{UpdateAtoms}(\mathcal{M})$  is also a repair for  $\langle \mathcal{I}, \eta \rangle$ . Therefore, we have to prove that  $\text{UpdateAtoms}(\mathcal{M})$  is founded.

Let  $\alpha \in \text{UpdateAtoms}(\mathcal{M})$ ,  $\text{ground}(\mathcal{FP}(\eta))$  contains the constraint  $\leftarrow \alpha$ ,  $\text{not } \alpha^f$ , thus  $\mathcal{M}$  contains the atom  $\alpha^f$ . As  $\alpha^f$  is supported,  $\text{ground}(\mathcal{FP}(\eta))$  contains a rule  $\rho = \alpha^f \leftarrow \Phi$ , with  $\Phi$  of the form

$$\begin{aligned}
\bigwedge_{i=1}^{n-1} ((L_i \wedge \text{not } \text{comp}^{-1}(L_i)) \vee \text{comp}^{-1}(\text{not } L_i)), \\
(\text{comp}(\alpha) \vee \text{comp}^{-1}(\text{not } \text{comp}(\alpha)))
\end{aligned}$$

such that  $\mathcal{M} \models \Phi$ . As  $\Phi$  does not contain any atom in the form  $\pm a^f(t)$ ,  $\mathcal{M}^r \models \Phi$ . The rule  $\rho$  belongs to a set  $\text{fp}(r)$ , where  $r$  is a ground active integrity constraint belonging to  $\text{ground}(\eta)$  and is in the form

$$\bigwedge_{i=1}^{n-1} L_i, \text{comp}(\alpha) \supset \bigvee_{j=1}^{p-1} \alpha_j \vee \alpha.$$

Observe that  $\Phi$  is *true* iff  $\text{body}(r)$ , evaluated over the database  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\})$ , is *true*. Thus, as  $(\text{UpdateAtoms}(\mathcal{M}) \cup \mathcal{I}) \models \text{body}(\rho)$ , it follows that  $\mathcal{I} \circ (\text{UpdateAtoms}(\mathcal{M}) \setminus \{\alpha\}) \models \text{body}(r)$ , that is the atom  $\alpha$  appearing in the head of  $r$  is founded.

- (Completeness) Now, we prove that  $\mathbf{FR}(\mathcal{I}, \eta) \subseteq \text{UpdateAtoms}(\mathcal{SM}(\mathcal{FDP}(\eta) \cup \mathcal{I}))$ , that is that for each  $\mathcal{R} \in \mathbf{FR}(\mathcal{I}, \eta)$  there is a stable model  $\mathcal{M} \in \mathcal{SM}(\mathcal{FDP}(\eta) \cup \mathcal{I})$  such that  $\text{UpdateAtoms}(\mathcal{M}) = \mathcal{R}$ .



Let  $\mathcal{R} \in \mathbf{FR}(\mathcal{I}, \eta)$ . As  $\mathcal{R}$  is a repair,  $\mathcal{R} \cup \mathcal{I}$  is a stable model of  $\mathcal{DP}(ic(\eta)) \cup \mathcal{I}$  (Theorem (5.6)). We show that  $\mathcal{M} = \mathcal{R} \cup \mathcal{I} \cup \{\alpha^f \mid \alpha \in \mathcal{R}\}$  is a stable model for  $\mathcal{FDP}(\eta) \cup \mathcal{I}$ , that is that for every  $A \in \mathcal{R}$  there is a rule  $\rho$  in  $ground(\mathcal{FDP}(\eta))$  such that  $head(\rho) = \{\alpha^f\}$  and  $(\mathcal{R} \cup \mathcal{I}) \models body(\rho)$ .

Let  $r \in ground(\eta)$  a ground active integrity constraint supporting the update action  $\alpha \in \mathcal{R}$  of the form

$$\bigwedge_{i=1}^{n-1} L_i, comp(A) \supset \bigvee_{j=1}^{p-1} \alpha_j \vee \alpha$$

We have that  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\}) \models body(r)$  that is  $\mathcal{I} \circ \mathcal{R} \models \bigwedge_{i=1}^{n-1} L_i$ , for  $i \in [1..n-1]$ , and  $\mathcal{I} \circ (\mathcal{R} \setminus \{\alpha\}) \models comp(\alpha)$ . This means that if  $A$  is in the form  $-a(x)$ ,  $a(x) \in \mathcal{I}$  or  $+a(x) \in \mathcal{R}$  whereas, if  $\alpha$  is in the form  $+a(x)$ ,  $a(x) \notin \mathcal{I}$  or  $-a(x) \in \mathcal{R}$ . Moreover, for each positive literal  $L_i = l(y)$  ( $i \in [1..n-1]$ ),  $l(y) \in \mathcal{I}$  and  $-l(y) \notin \mathcal{R}$  or  $+l(y) \in \mathcal{R}$  whereas, for each negative literal  $L_i = not\ l(y)$  ( $i \in [1..n-1]$ ),  $l(y) \notin \mathcal{I}$  and  $+l(y) \notin \mathcal{R}$  or  $-l(y) \in \mathcal{R}$ .

Therefore, the conjunction  $\Phi = \bigwedge_{i=1}^n ((L_i \wedge not\ comp^{-1}(L_i)) \vee comp^{-1}(not\ L_i))$ ,  $(comp(\alpha) \vee comp^{-1}(not\ comp(\alpha)))$  is true in  $\mathcal{R} \cup \mathcal{I}$ .

Moreover, the program  $ground(\mathcal{FDP}(\eta))$  contains a rule of the form  $\alpha^f \leftarrow \Phi$  whose body is true in  $\mathcal{R} \cup \mathcal{I}$ . This is exactly the rule  $\rho$  supporting  $\alpha^f$ .  $\square$

*Example 3.31.* Consider again Example 3.29. The unique stable model of  $\mathcal{FDP}(\eta) \cup \mathcal{I}$  is  $\mathcal{M} = \{p(a), p(b), q(a), -p(a), -p^f(a)\}$  corresponding to the founded repair  $\mathcal{R} = \{-p(a)\}$ .  $\square$

### 3.4.2 Data Complexity

Given a (standard) integrity constraint  $r$  of the form 3.2,  $Ext(r)$  denotes the AIC

$$(\forall X) \left[ \bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n not\ b_j(X_j), \varphi(X_0) \supset \bigvee_{j=1}^m -b_j(X_j) \vee \bigvee_{j=m+1}^n +b_j(X_j) \right]$$

The definition of  $Ext(\cdot)$  is extended to sets of AICs in the standard way.

**Theorem 3.32.** *Let  $\mathcal{I}$  be a database and  $\eta$  a set of active integrity constraints. The problem of deciding whether there exists a founded repair for  $\langle \mathcal{I}, \eta \rangle$  is  $\Sigma_2^P$ -complete.*

**Proof.**

*Membership.* In [54] it has been shown that the problem of deciding whether there exists a stable model for a disjunctive Datalog program is  $\Sigma_2^P$ -complete. As every founded repair can be derived from a stable model of the disjunctive Datalog program  $\mathcal{FDP}(\eta) \cup DB$ , the problem consists in checking whether there exists a stable model for  $\mathcal{FDP}(\eta) \cup DB$ .

*Hardness.* In [70] it has been shown that, for a given database  $\mathcal{I}$ , a set of standard integrity constraints  $\eta$  and set of update constraints  $\mathcal{UC}$ , the problem of checking if there exists a repair  $\mathcal{R}$  for  $\langle \mathcal{I}, \eta \rangle$  such that every update constraint in  $\mathcal{UC}$  is satisfied is  $\Sigma_2^p$ -complete. An update constraint is of one of the following two forms

$$\begin{aligned} &\leftarrow \text{insert}(q(t_1, \dots, t_n)) \\ &\leftarrow \text{delete}(q(t_1, \dots, t_n)) \end{aligned}$$

and states that every atom  $q(u_1, \dots, u_n)$  unifying with  $q(t_1, \dots, t_n)$  cannot be inserted or deleted, respectively.

Consider the set of active integrity constraints  $\eta' = \text{ground}(\text{Ext}(\eta))$  and the set  $\eta''$  obtained by deleting from the head of rules in  $\eta'$  all update actions of the form  $+a(t)$  such that  $\leftarrow \text{insert}(a(t))$  is in  $\text{ground}(\mathcal{UC})$  (resp.  $-a(t)$  such that  $\leftarrow \text{delete}(a(t))$  is in  $\text{ground}(\mathcal{UC})$ ). Clearly, the set of ground constraints  $\eta''$  is equivalent to  $\text{ground}(\text{Ext}(\eta)) \cup \text{ground}(\mathcal{UC})$  as it is obtained by deleting from the head of the active integrity constraints in  $\text{ground}(\text{Ext}(\eta))$  update actions which cannot be derived. After the deletion of the useless head update actions, update constraints are not necessary any more and the problem consists in deciding whether there exists a founded repair for  $\mathcal{I}(\eta'')$ .  $\square$

The consistent founded answer to a relational query  $Q = (g, \mathcal{P})$  over a database  $\mathcal{I}$  with active integrity constraints  $\eta$  (denoted by  $Q(\mathcal{I}, \eta)$ ), is obtained by first computing the set  $\mathbf{FR}(\mathcal{I}, \eta)$  of founded repairs for  $\mathcal{I}$  and, then, considering the intersection  $\bigcap_{\mathcal{R} \in \mathbf{FR}(\mathcal{I}, \eta)} Q(\mathcal{I} \circ \mathcal{R})$ .

**Theorem 3.33.** *Let  $\mathcal{I}$  be a database and  $\eta$  a set of active integrity constraints. The problem of deciding whether a ground atom  $g(t)$  belongs to all repaired databases obtained by means of founded repairs for  $\langle \mathcal{I}, \eta \rangle$  is  $\Pi_2^p$ -complete.*

**Proof.**

*Membership.* As every founded repair for  $\langle \mathcal{I}, \eta \rangle$  can be derived from a stable model of  $\mathcal{FDP}(\eta) \cup \mathcal{I}$ , it is sufficient to check that the ground atom  $g'(t)$  belongs to each stable model of the disjunctive Datalog program

$$\mathcal{FDP}(\eta) \cup \mathcal{I} \cup \{g(t)' \leftarrow (g(t) \wedge \text{not } -g(t)) \vee +g(t)\}.$$

This is a well known  $\Pi_2^p$ -complete problem.

*Hardness.* In [70] it has also been shown that, for a given database  $\mathcal{I}$ , a set of full (standard) integrity constraints  $\eta$  and set of update constraints  $\mathcal{UC}$ , the problem of checking if all repaired databases contain an atom  $g(t)$  is  $\Pi_2^p$ -complete.

In the proof of Theorem 5.28 it has been shown that for every set of full (standard) integrity constraints  $\eta$  and set of update constraints  $\mathcal{UC}$  there exists a set of “equivalent” active integrity constraints  $\eta''$ , that is for every database  $\mathcal{I}$  the set of repairs satisfying  $\eta$  and  $\mathcal{UC}$  is equivalent to the set of founded repairs satisfying  $\eta''$ . There-

fore, the problem of checking whether a ground atom  $g(t)$  belongs to all repaired databases obtained by means of founded repairs is  $\Pi_2^P$ -hard.  $\square$

In a similar way it is possible to prove that for not disjunctive active integrity constraints the complexity is in the first level of the polynomial hierarchy.

### 3.4.3 Preferred Repairs and Answers

A founded repair for a set of active integrity constraints is not guaranteed to exist. Nevertheless, it is often necessary to provide a repair, even if no founded repair exists, or to compute consistent answers to queries. Thus, in this section we define an approach that always permits us to obtain a consistent repaired database. In particular, we interpret the actions in the head of constraints as an indication of the operations the user prefers to perform to make the database consistent. Moreover, as the presence of existentially quantified variables, could produce a possibly infinite number of repairs, we only consider universally quantified active integrity constraints. Firstly, we introduce a partial order on the repairs.

**Definition 3.34.** *Let  $\mathcal{I}$  be a database,  $\eta$  a set of active integrity constraints and  $\mathcal{R}_1, \mathcal{R}_2$  two repairs for  $\langle \mathcal{I}, \eta \rangle$ . Then,  $\mathcal{R}_1$  is preferable to  $\mathcal{R}_2$  ( $\mathcal{R}_1 \sqsupseteq \mathcal{R}_2$ ) if  $Unfounded(\mathcal{R}_1, \mathcal{I}, \eta) \subset Unfounded(\mathcal{R}_2, \mathcal{I}, \eta)$ . A set of update actions  $\mathcal{R}$  is a preferred repair for  $\langle \mathcal{I}, \eta \rangle$  if it is a repair for  $\langle \mathcal{I}, \eta \rangle$  and there is no repair  $\mathcal{R}'$  for  $\langle \mathcal{I}, \eta \rangle$  such that  $\mathcal{R}' \sqsupseteq \mathcal{R}$ .  $\square$*

*Example 3.35.* Consider the integrity constraint of Example 3.2 with the database  $\mathcal{I} = \{mgr(john, b, 1000), mgr(franks, b, 2000), mgr(mary, c, 1000), mgr(rosy, c, 2000)\}$ . There are four repairs  $\mathcal{R}_1 = \{-mgr(john, b, 1000), -mgr(mary, c, 1000)\}$ ,  $\mathcal{R}_2 = \{-mgr(john, b, 1000), -mgr(rosy, c, 2000)\}$ ,  $\mathcal{R}_3 = \{-mgr(franks, b, 2000), -mgr(mary, c, 1000)\}$  and  $\mathcal{R}_4 = \{-mgr(franks, b, 2000), -mgr(rosy, c, 2000)\}$ . The order relation is  $\mathcal{R}_2 \sqsupseteq \mathcal{R}_1$ ,  $\mathcal{R}_3 \sqsupseteq \mathcal{R}_1$ ,  $\mathcal{R}_4 \sqsupseteq \mathcal{R}_2$  and  $\mathcal{R}_4 \sqsupseteq \mathcal{R}_3$ . Therefore, we have only one preferred repair which is also founded (namely  $\mathcal{R}_4$ ). Assume now we also have the constraint

$$not\ mgr(rosy, c, 2000) \supseteq$$

declaring that the tuple  $mgr(rosy, c, 2000)$  must be in  $\mathcal{I}$ . In such a case we only have the two repairs  $\mathcal{R}_1$  and  $\mathcal{R}_3$  and the preferred one is  $\mathcal{R}_3$  which is not founded.  $\square$

The relation  $\sqsupseteq$  is a *partial order* as it is irreflexive, antisymmetric and transitive. The set of all preferred repairs for a database  $\mathcal{I}$  and a set of active integrity constraints  $\eta$  is denoted by  $\mathbf{PR}(\mathcal{I}, \eta)$ .

Clearly, the relation between preferred, founded and standard repairs is as follows:  $\mathbf{FR}(\mathcal{I}, \eta) \subseteq \mathbf{PR}(\mathcal{I}, \eta) \subseteq \mathbf{R}(\mathcal{I}, \eta)$ . The next proposition states the precise relation between preferred, founded and general repairs.

**Fact 3.36** *Let  $\mathcal{I}$  be a database and  $\eta$  a set of AICs. If  $\mathbf{FR}(\mathcal{I}, \eta) \neq \emptyset$  then  $\mathbf{PR}(\mathcal{I}, \eta) = \mathbf{FR}(\mathcal{I}, \eta)$   $\square$*

Obviously, as the existence of a repair is guaranteed, the existence of a preferred repair is guaranteed too. We conclude by presenting a result on the computational complexity of computing preferred repairs and answers.

**Theorem 3.37.** *Let  $\mathcal{I}$  be a database and  $\eta$  a set of active integrity constraints, then*

1. *deciding whether there exists a preferred repair for  $\langle \mathcal{I}, \eta \rangle$  is  $\Sigma_2^p$ -complete;*
2. *deciding whether a ground atom  $g(t)$  belongs to all repaired databases obtained by means of preferred repairs is  $\Pi_2^p$ -complete.*

**Proof.**

1. *Membership.* From Theorem 3.36 to check if  $\mathbf{PR}(\mathcal{I}, \eta) \neq \emptyset$  it is sufficient to check if  $\mathbf{R}(\mathcal{I}, \eta) \neq \emptyset$  (which is a  $\Sigma_2^p$ -complete problem).

*Hardness.* Consider the database  $\mathcal{I}' = \mathcal{I} \cup \{a\}$  and the set of constraints  $\eta' = \eta \cup \{a \supset\}$  where  $a$  is a new atom not appearing in  $\mathcal{I}$ . The problem of deciding whether  $\mathbf{R}(\mathcal{I}, \eta) \neq \emptyset$  (which is  $\Sigma_2^p$ -complete) is equivalent to the problem of deciding whether  $\mathbf{PR}(\mathcal{I}', \eta') \neq \emptyset$ , as  $\langle \mathcal{I}', \eta' \rangle$  does not admit a founded repair.

2. *Membership.* To decide whether  $g(t) \in \bigcap_{\mathcal{R} \in \mathbf{PR}(\mathcal{I}, \eta)} \mathcal{I} \circ \mathcal{R}$  it is sufficient to checking whether  $g(t) \in \bigcap_{\mathcal{R} \in \mathbf{FR}(\mathcal{I}, \eta)} \mathcal{I} \circ \mathcal{R}$  if  $\mathbf{FR}(\mathcal{I}, \eta) \neq \emptyset$  and to deciding whether  $g(t) \in \bigcap_{\mathcal{R} \in \mathbf{R}(\mathcal{I}, \eta)} \mathcal{I} \circ \mathcal{R}$  if  $\mathbf{FR}(\mathcal{I}, \eta) = \emptyset$  (both problems are  $\Pi_2^p$ -complete).

*Hardness.* The problem of deciding whether  $g(t) \in \bigcap_{\mathcal{R} \in \mathbf{R}(\mathcal{I}, \eta)} \mathcal{I} \circ \mathcal{R}$  (which is  $\Pi_2^p$ -complete) is equivalent to the problem of deciding whether  $g(t) \in \bigcap_{\mathcal{R} \in \mathbf{PR}(\mathcal{I}', \eta')} \mathcal{I} \circ \mathcal{R}$ , where  $\mathcal{I}'$  and  $\eta'$  are the ones used in the proof of Part (1).  $\square$

The above theorem states that computing preferred repairs and answers is not harder than computing standard or founded repairs and answers.

## Active Integrity Constraints and Revision Programming

**Summary.** We compare *active integrity constraints* [29] and *revision programming* [108], two formalisms designed to describe integrity constraints on databases and to specify *preferred* ways to enforce them. The original semantics proposed for these formalisms differ. The semantics for active integrity constraints defines the concept of *founded repair*. Intuitively, a founded repair is a minimal set of *update actions* (*insertions* and *deletions*), defined and supported by active integrity constraints, to be performed over the database in order to make it consistent. The semantics for revision programs defines the concept of *justified revision*. A justified revision is a set of *revision literals*, an alternative way to model updates over a database, that can be inferred by means of the revision program and by the set of all atoms that do not change their state of *presence* (**in**) or *absence* (**out**) during the update process. We show that each founded repair corresponds to a justified revision, but not vice-versa. We introduce two new semantics: one for active integrity constraints and one for revision programs. The first one allows us to compute a smaller set of repairs, the *justified repairs*, that correspond to justified revisions. The second one allow us to compute a wider set of revision, the *founded revisions*, that correspond to founded repairs. The introduction of these new semantics for the two formalisms shows that each of them can be ported to the other one, and that once it is done, both frameworks become equivalent under a certain simple syntactic transformation. We show that for each semantics the *shifting property* holds. Shifting consists of transforming an instance of a database repair problem to another syntactically isomorphic instance by changing active integrity constraints or revision programs to reflect the “shift” from the original database to the new one.

### 4.1 Introduction

Active integrity constraints *explicitly* encode both integrity constraints and preferred way to enforce them in the case they are violated. To specify a precise meaning of sets of active integrity constraints in the previous chapter the concept of *founded repair* has been presented. Founded repairs are change-minimal and satisfy a certain groundedness condition.

Revision programs consist of *revision rules*. Each revision rule represents an integrity constraint, and *implicitly* encodes preferred ways to enforce it by means of a certain

syntactic convention. Following intuitions from logic programming, [108] proposed two semantics for revision programs: the semantics of *justified* revisions and the semantics of *supported* revisions. Each semantics reflects preferences on the ways to repair a database with respect to a revision program. In general, neither semantics satisfies the minimality of change principle. Justified revisions generalize the answer set semantics of Lifschitz-Woo programs [100].

#### 4.1.1 Contribution

The original semantics of active integrity constraints and revision programming seemingly cannot be related in any direct way. They have different computational properties. For instance, the problem of the existence of a founded repair for normal active integrity constraints is  $\Sigma_P^2$ -complete, while the same problem for justified revisions of normal revision programs is NP-complete. Furthermore, the semantics for revision programming do not have the minimality of change property, while founded repairs with respect to active integrity constraints do.

In this chapter, we demonstrate that despite the differences in the syntax, and the lack of a simple correspondence between justified revisions and founded repairs, the formalisms of revision programs and active integrity constraints are closely related. There are two keys to the relationship. First, we need a certain syntactic restriction on revision programs. Specifically, we introduce the class of *proper* revision programs and show that restricting to proper programs does not affect the expressive power.

Second, we need to broaden the families of the semantics for each formalism so that the two sides could be aligned. To this end for active integrity constraints we introduce new semantics by dropping the minimality of change condition, which results in the semantics of *weak repairs* and *founded weak repairs*. We also adapt to the case of active integrity constraints the semantics of justified revisions (justified weak revisions), which leads us to the semantics of *justified weak repairs* and *justified repairs*. For revision programs, we modify the semantics of revisions and justified revisions by imposing on them the minimality condition. Moreover we introduce the semantics of *founded revisions* (*founded weak revisions*) that corresponds to the semantics of founded repairs (founded weak repairs). We show that under a simple bijection between proper revision programs and active integrity constraints, founded (weak) revisions correspond to founded (weak) repairs and justified (weak) revisions correspond to justified (weak) repairs. This result demonstrates that both formalisms, even though rooted in different intuitions, can be “completed” so that to become notational variants of each other.

Both in the case of active integrity constraints and revision programs, the concepts of “groundedness” we consider do not imply, in general, the property of the minimality of change. However, in each case, there are theories when it is the case. We present two broad classes of sets of active integrity constraints (revision programs, respectively) for which groundedness based on the notion of being justified implies minimality.

A fundamental property of semantics describing database updates is the invariance under *shifting* (we introduce it formally later in the chapter). The semantics of revision programming have this property [108, 118]. In this chapter we extend it to the semantics of active integrity constraints.

### 4.1.2 Plan of the Chapter

The chapter is organized as follows. In the next section, we introduce the concepts of weak repairs and founded weak repairs. In the following section we present justified weak repairs and justified repairs. We discuss the *normalization* of active integrity constraints in Section 4.4. It leads to an additional semantics for active integrity constraints, arguably best grounded in an initial database and active integrity constraints. In Section 4.5, we establish the invariance under shifting for the semantics of active integrity constraints. We then study the complexity of problems for the semantics of justified (weak) repairs.

Next, we recall basic concepts of revision programming. We then introduce some new semantics for revision programs, and show that they are invariant under shifting. In the main result of this section, we establish a precise connection between active integrity constraints and revision programs.

## 4.2 Weak Repairs and Founded Weak Repairs

For the sake of simplicity, in this chapter we will discuss only the propositional case. However, definitions and results can be lifted to the predicate case. We consider a finite set  $At$  of propositional atoms. We represent databases as subsets of  $At$ . The concept of weak repair is obtained by removing the minimality of change property from the concept of repair (Definition 3.3).

**Definition 4.1.** (WEAK REPAIR) *Let  $\mathcal{I}$  be a database and  $\eta$  a set of integrity constraints. A weak repair for  $\langle \mathcal{I}, \eta \rangle$  is a consistent set  $\mathcal{U}$  of update actions such that*

- $(\{+a \mid a \in \mathcal{I}\} \cup \{-a \mid a \in At \setminus \mathcal{I}\}) \cap \mathcal{U} = \emptyset$   
( $\mathcal{U}$  consists of “essential” update actions only), and
- $\mathcal{I} \circ \mathcal{U} \models \eta$  (constraint enforcement). □

Observe that the first property in the previous definition is not imposed explicitly in Definition 3.3 as it is ensured by the minimality of change.

Most applications require the minimality of change. Thus, for the most part, we are interested in properties of repairs. However, weak repairs have also interesting properties and offer a broader perspective. Therefore, in this chapter we consider them explicitly.

We recall that a (*ground*) *active integrity constraint* is an expression of the form

$$r = L_1, \dots, L_m \supset \alpha_1 | \dots | \alpha_k \quad (4.1)$$

where  $L_i$  are literals such that  $L_h \neq L_k^D$  for each  $h \neq k$ ,  $\alpha_j$  are update actions, and

$$\{lit(\alpha_1)^D, \dots, lit(\alpha_k)^D\} \subseteq \{L_1, \dots, L_m\} \quad (4.2)$$

The role of the condition (4.2) is to ensure that an active integrity constraint supports only those update actions that can “fix” it (executing them ensures that the resulting database satisfies the constraint). The condition can be stated concisely as follows:  $[lit(head(r))]^D \subseteq body(r)$ . We call literals in  $[lit(head(r))]^D$  *updatable* by  $r$ . They are precisely those literals that can be affected by an update action in  $head(r)$ . We call every literal in  $body(r) \setminus [lit(head(r))]^D$  *non-updatable* by  $r$ . We denote the set of literals updatable by  $r$  as  $up(r)$  and the set of literals non-updatable by  $r$  as  $nup(r)$ .

To formalize the notion of “support” and translate it into a method to select “preferred” repairs, in the previous chapter we proposed the concept of a *founded repair* — a repair that is *founded* (in some sense, *implied*) by a set of active integrity constraints. Now we introduce an alternative definition of founded repair which is equivalent to Definition 3.13 but more useful for the proofs presented in this chapter. In addition, we introduce the semantics of *founded weak repairs*.

**Definition 4.2.** (FOUNDED (WEAK) REPAIR) *Let  $\mathcal{I}$  be a database,  $\eta$  a set of active integrity constraints, and  $\mathcal{U}$  a consistent set of update actions.*

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

We observe that the foundedness does not imply the constraint enforcement nor the minimality of change. Let  $\mathcal{I} = \emptyset$  and  $\eta$  consist of the following active integrity constraints:

$$\begin{aligned} r_1 &= not\ a \supset +a \\ r_2 &= not\ b, c \supset +b \\ r_3 &= b, not\ c \supset +c. \end{aligned}$$

The unique founded repair for  $\langle \mathcal{I}, \eta \rangle$  is  $\{+a\}$ . The set  $\{+a, +b, +c\}$  is founded, guarantees constraint enforcement (and so, it is a founded weak repair), but it is *not* change-minimal. The set  $\{+b, +c\}$  is founded but does not guarantee constraint enforcement. Therefore, in the definition of founded (weak) repairs, the property of being a (weak) repair must be enforced explicitly. We also note that foundedness properly narrows down the class of repairs. If  $\eta = \{a, b \supset -b\}$ , and  $\mathcal{I} = \{a, b\}$  (an example we considered earlier),  $\mathcal{U} = \{-a\}$  is a repair for  $\langle \mathcal{I}, \eta \rangle$  but not a founded repair.



In some cases, founded repairs, despite combining foundedness with change-minimality, are still not grounded strongly enough. The problem is the circularity of support.

*Example 4.3.* Let  $\mathcal{I} = \{a, b\}$  and  $\eta$  consist of the following active integrity constraints:

$$\begin{aligned} r_1 &= a, b \quad \supset \neg a \mid \neg b \\ r_2 &= a, \text{ not } b \quad \supset \neg a \\ r_3 &= \text{ not } a, b \quad \supset \neg b. \end{aligned}$$

We note that  $\mathcal{U} = \{-a, -b\}$  is a founded repair for  $\langle \mathcal{I}, \eta \rangle$ . Indeed,  $\neg a$  is founded with respect to  $\langle \mathcal{I}, \eta \rangle$  and  $\mathcal{U}$ , with  $r_2$  providing the necessary support. Similarly,  $\neg b$  is founded with respect to  $\langle \mathcal{I}, \eta \rangle$  and  $\mathcal{U}$  because of  $r_3$ .

The active integrity constraint  $r_1$  is the only constraint violated by  $\mathcal{I}$  and so, it is the one that forces the need for a repair. However,  $r_1$  itself provides no support for  $\neg a$  or  $\neg b$ . It follows that the support for foundedness of  $\neg a$  is provided solely by  $r_2$  and it *requires* that  $\neg b$  be included in the repair. Similarly, the support for foundedness of  $\neg b$  is provided solely by  $r_3$  and it depends on  $\neg a$  being included in the repair. Thus, the foundedness of  $\{-a, -b\}$  is “circular”:  $\neg a$  is founded (and so included in  $\mathcal{U}$ ) due to the fact that  $\neg b$  has been included in  $\mathcal{U}$ , and  $\neg b$  is founded (and so included in  $\mathcal{U}$ ) due to the fact that  $\neg a$  has been included in  $\mathcal{U}$ .  $\square$

To summarize this section, the semantics of founded repairs gives preference to some ways of repairing active integrity constraints over others. It only considers repairs whose all elements are founded. However, foundedness may be circular and so the associated concept of groundedness is weak. We revisit this issue in the next section.

On the computational side, the complexity of the semantics of repairs is lower than that of founded repairs. The problem of the existence of a repair is NP-complete, while the problem of the existence of a founded repair is  $\Sigma_P^2$ -complete (see previous chapter). For the sake of completeness, we also note that the problem of the existence of a founded weak repair is again “only” NP-complete (the proof is simple and we omit it).

### 4.3 Justified Repairs

In this section, we will introduce another semantics for active integrity constraints that captures a stronger concept of groundedness than the one behind founded repairs. The goal is to disallow circular dependencies like the one we discussed in Example 4.3.

We start by defining when a set of update actions is *closed* under active integrity constraints. Let  $\eta$  be a set of active integrity constraints and let  $\mathcal{U}$  be a set of update actions. If  $r \in \eta$ , and for every *non-updatable* literal  $L \in \text{body}(r)$  there is an update action  $\alpha \in \mathcal{U}$  such that  $\text{lit}(\alpha) = L$  then, after applying  $\mathcal{U}$  or any of its consistent supersets to the initial database, the result of the update, say  $\mathcal{R}$ , satisfies all non-updatable literals in  $\text{body}(r)$ . To guarantee that  $\mathcal{R}$  satisfies  $r$ ,  $\mathcal{R}$  must *falsify* at least

one literal in  $body(r)$ . To ensure that  $\mathcal{U}$  should contain at least one update action from  $head(r)$ .

**Definition 4.4.** (CLOSED SETS OF UPDATE ACTIONS)

A set  $\mathcal{U}$  of update actions is closed under an active integrity constraint  $r$  if  $nup(r) \not\subseteq lit(\mathcal{U})$ , or  $head(r) \cap \mathcal{U} \neq \emptyset$ .

A set  $\mathcal{U}$  of update actions is closed under a set  $\eta$  of active integrity constraints if it is closed under every  $r \in \eta$ .  $\square$

If a set of update actions is not closed under a set  $\eta$  of active integrity constraints, executing its elements is not guaranteed to enforce constraints represented by  $\eta$ . Therefore closed sets of update actions are important. We regard *minimal* such sets as “forced” by  $\eta$ , as all elements in a minimal set of update actions closed under  $\eta$  are necessary (no nonempty subset can be dropped).

Another key notion in our considerations is that of *no-effect actions*. Let  $\mathcal{I}$  be a database and  $\mathcal{R}$  a result of updating  $\mathcal{I}$ . An update action  $+a$  (respectively,  $-a$ ) is a *no-effect* action with respect to  $(\mathcal{I}, \mathcal{R})$  if  $a \in i \cap \mathcal{R}$  (respectively,  $a \notin i \cup \mathcal{R}$ ). We denote by  $ne(\mathcal{I}, \mathcal{R})$  the set of all no-effect actions with respect to  $(\mathcal{I}, \mathcal{R})$ . We note the following two simple properties.

**Proposition 4.5.** Let  $\mathcal{I}$  be a database. Then

1. For every database  $\mathcal{R}$ ,  $\mathcal{R} \circ ne(\mathcal{I}, \mathcal{R}) = \mathcal{R}$
2. For every set  $\mathcal{E}$  of update actions such that  $\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  is consistent,  $\mathcal{I} \circ \mathcal{E} = \mathcal{I} \circ (\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}))$ .

**Proof:** (1) Since  $ne(\mathcal{I}, \mathcal{R}) = \{+a \mid a \in i \cap \mathcal{R}\} \cup \{-a \mid a \notin i \cup \mathcal{R}\}$ ,  $\mathcal{R} \circ ne(\mathcal{I}, \mathcal{R}) = (\mathcal{R} \cup (\mathcal{I} \cap \mathcal{R})) \cap (\mathcal{I} \cup \mathcal{R}) = \mathcal{R}$ .

(2) Since  $\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  is consistent, Proposition 4.5 imply that  $\mathcal{I} \circ (\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})) = (\mathcal{I} \circ \mathcal{E}) \circ ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \mathcal{I} \circ \mathcal{E}$ .  $\square$

Our semantics of justified repairs is based on the knowledge-representation principle (a form of the frame axiom) that remaining in the previous state requires no reason (persistence by inertia). Thus, when justifying update actions necessary to transform  $\mathcal{I}$  into  $\mathcal{R}$  based on  $\eta$  we assume the set  $ne(\mathcal{I}, \mathcal{R})$  as given. This brings us to the notion of a justified weak repair.

**Definition 4.6.** (JUSTIFIED WEAK REPAIRS)

Let  $\mathcal{I}$  be a database and  $\eta$  a set of active integrity constraints. A consistent set  $\mathcal{U}$  of update actions is a justified action set for  $\langle \mathcal{I}, \eta \rangle$  if  $\mathcal{U}$  is a minimal set of update actions containing  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$  and closed under  $\eta$ .

If  $\mathcal{U}$  is a justified action set for  $\langle \mathcal{I}, \eta \rangle$ , then  $\mathcal{E} = \mathcal{U} \setminus ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$  is a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ .  $\square$

Intuitively, a set  $\mathcal{U}$  of update actions is a justified action set, if it is precisely the set of update actions forced or *justified* by  $\eta$  and the no-effect actions with respect to  $\mathcal{I}$

and  $\mathcal{I} \circ \mathcal{U}$ . This “fixpoint” aspect of the definition is reminiscent of the definitions of semantics of several nonmonotonic logics, including (disjunctive) logic programming with the answer set semantics. The connection can be made more formal and we take advantage of it in the section on the complexity and computation.

We will now study justified action sets and justified weak repairs. We start with an alternative characterization of justified weak repairs.

**Theorem 4.7.** *Let  $\mathcal{I}$  be a database,  $\eta$  a set of active integrity constraints and  $\mathcal{E}$  a consistent set of update actions. Then  $\mathcal{E}$  is a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$  if and only if  $\mathcal{E} \cap ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \emptyset$  and  $\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  is a justified action set for  $\langle \mathcal{I}, \eta \rangle$ .*

**Proof:** ( $\Rightarrow$ ) Since  $\mathcal{E}$  is a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ ,  $\mathcal{E} = \mathcal{U} \setminus ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$  for some consistent set  $\mathcal{U}$  of update actions such that  $\mathcal{U}$  is minimal containing  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$  and closed under  $\eta$ . By Proposition 4.5(2),  $\mathcal{I} \circ \mathcal{U} = \mathcal{I} \circ \mathcal{E}$ . Thus,  $\mathcal{E} \cap ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \emptyset$ . Moreover, since  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U}) \subseteq \mathcal{U}$ ,  $\mathcal{U} = \mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . Hence,  $\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  is a justified action set for  $\langle \mathcal{I}, \eta \rangle$ .

( $\Leftarrow$ ) Let  $\mathcal{U} = \mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . We will show that  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U}) = ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . To this end, let  $+a \in ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ . Then,  $a \in i$  and  $-a \notin \mathcal{U}$  (the latter property follows by the consistency of  $\mathcal{U}$ ). It follows that  $-a \notin \mathcal{E}$  and, consequently,  $+a \in ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . Similarly, we show that if  $-a \in ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ , then  $-a \in ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . Thus, we obtain that  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U}) \subseteq ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ .

Conversely, let  $+a \in ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . Then  $a \in i$  and  $+a \in \mathcal{U}$ . Since  $\mathcal{U}$  is consistent (it is a justified action set for  $\langle \mathcal{I}, \eta \rangle$ ),  $\mathcal{I} \circ \mathcal{U}$  is well defined and  $+a \in ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ . The case  $-a \in ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  is similar. Thus,  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) \subseteq ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$  and the claim follows.

Since  $\mathcal{E} \cap ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \emptyset$ , we obtain that  $\mathcal{E} = \mathcal{U} \setminus ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ . Since  $\mathcal{U}$  is a justified action set for  $\langle \mathcal{I}, \eta \rangle$ ,  $\mathcal{E}$  is a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ .  $\square$

Justified weak repairs have two key properties for the problem of database update: constraint enforcement (hence the term “weak repair”) and foundedness.

**Theorem 4.8.** *Let  $\mathcal{I}$  be a database,  $\eta$  a set of active integrity constraints, and  $\mathcal{E}$  a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ . Then*

1. *For every atom  $a$ , exactly one of  $+a$  or  $-a$  is in  $\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$*
2.  *$\mathcal{I} \circ \mathcal{E} \models \eta$*
3.  *$\mathcal{E}$  is founded for  $\langle \mathcal{I}, \eta \rangle$ .*

**Proof:** Throughout the proof, use the notation  $\mathcal{U} = \mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ .

(1) Since  $\mathcal{U}$  is consistent (cf. Theorem 4.7), for every atom  $a$ , at most one of  $+a$ ,  $-a$  is in  $\mathcal{U}$ . If  $+a \in ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  or  $-a \in ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  then the claim follows. Otherwise, the status of  $a$  changes as we move from  $\mathcal{I}$  to  $\mathcal{I} \circ \mathcal{E}$ . That is, either  $+a$  or  $-a$  belongs to  $\mathcal{E}$  and, consequently, to  $\mathcal{U}$ , as well.

(2) Let us consider  $r \in \eta$ . Since  $\mathcal{U}$  is closed under  $\eta$  (cf. Theorem 4.7), we have  $nup(r) \not\subseteq lit(\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}))$  or  $head(r) \cap (\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})) \neq \emptyset$ . Let us assume

the first possibility, and let  $L$  be a literal such that  $L \in nup(r)$  and  $ua(L) \notin \mathcal{U}$ . By (1),  $ua(L^D) \in \mathcal{U}$ . Consequently,  $\mathcal{I} \circ \mathcal{U} \not\models L$ . By Proposition 4.5(2),  $\mathcal{I} \circ \mathcal{E} \not\models L$ . Since  $L \in body(r)$ ,  $\mathcal{I} \circ \mathcal{E} \models r$ .

Thus, let us assume that  $head(r) \cap \mathcal{U} \neq \emptyset$  and let  $\alpha \in head(r) \cap \mathcal{U}$ . Then  $\alpha \in head(r)$  and so,  $lit(\alpha)^D \in body(r)$ . Furthermore,  $\alpha \in \mathcal{U}$  and so,  $\mathcal{I} \circ \mathcal{U} \models lit(\alpha)$ . By Proposition 4.5(2),  $\mathcal{I} \circ \mathcal{E} \models lit(\alpha)$ . Thus,  $\mathcal{I} \circ \mathcal{E} \models r$  in this case, too.

(3) Let  $\alpha \in \mathcal{E}$ . By Theorem 4.7,  $\alpha \notin ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . Thus,  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) \subseteq \mathcal{U} \setminus \{\alpha\}$ . Since  $\mathcal{U}$  is a minimal set closed under  $\eta$  and containing  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ ,  $\mathcal{U} \setminus \{\alpha\}$  is not closed under  $\eta$ . That is, there is  $r \in \eta$  such that  $nup(r) \subseteq lit(\mathcal{U} \setminus \{\alpha\})$  and  $head(r) \cap (\mathcal{U} \setminus \{\alpha\}) = \emptyset$ .

We have

$$\mathcal{I} \circ (\mathcal{U} \setminus \{\alpha\}) = \mathcal{I} \circ (ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) \cup (\mathcal{E} \setminus \{\alpha\})) = (\mathcal{I} \circ ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})) \circ (\mathcal{E} \setminus \{\alpha\}).$$

By Proposition 4.5 (and the fact that  $ne(\mathcal{I}, \mathcal{R}) = ne(\mathcal{R}, \mathcal{I})$ , for every databases  $\mathcal{I}$  and  $\mathcal{R}$ ),

$$\mathcal{I} \circ (\mathcal{U} \setminus \{\alpha\}) = \mathcal{I} \circ (\mathcal{E} \setminus \{\alpha\}). \quad (4.3)$$

From  $nup(r) \subseteq lit(\mathcal{U} \setminus \{\alpha\})$ , it follows that  $\mathcal{I} \circ (\mathcal{U} \setminus \{\alpha\}) \models nup(r)$ . By (4.3),  $\mathcal{I} \circ (\mathcal{E} \setminus \{\alpha\}) \models nup(r)$ . Since  $\alpha \in head(r)$ ,  $lit(\alpha^D) \notin nup(r)$ . Thus,  $\mathcal{I} \circ \mathcal{E} \models nup(r)$ .

The inclusion  $nup(r) \subseteq lit(\mathcal{U} \setminus \{\alpha\})$  also implies  $nup(r) \subseteq lit(\mathcal{U})$ . Since  $\mathcal{U}$  is closed under  $\eta$ ,  $head(r) \cap \mathcal{U} \neq \emptyset$  and so,  $head(r) \cap \mathcal{U} = \{\alpha\}$ .

Let us consider  $\beta \in head(r)$  such that  $\beta \neq \alpha$ . It follows that  $\beta \notin \mathcal{U}$ . By (1),  $\beta^D \in \mathcal{U}$  and, consequently,  $\mathcal{I} \circ \mathcal{U} \models \beta^D$ . Since  $\mathcal{I} \circ \mathcal{U} = \mathcal{I} \circ \mathcal{E}$  (Proposition 4.5), it follows that  $\alpha$  is founded with respect to  $\langle \mathcal{I}, \eta \rangle$  and  $\mathcal{E}$ .  $\square$

Theorem 4.8 directly implies that justified weak repairs are founded weak repairs.

**Corollary 4.9.** *Let  $\mathcal{I}$  be a database,  $\eta$  a set of active integrity constraints, and  $\mathcal{E}$  a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ . Then,  $\mathcal{E}$  is a founded weak repair for  $\langle \mathcal{I}, \eta \rangle$ .*

The converse to Corollary 4.9 does not hold. That is, there are founded weak repairs that are not justified weak repairs.

*Example 4.10.* Let  $\mathcal{I} = \{a, b\}$  and  $\eta$  consist of the following active integrity constraints:

$$\begin{aligned} r_1 &= a, b, not\ c \supset +c \vee -a \\ r_2 &= a, not\ b \supset -a \\ r_3 &= not\ a, b \supset -b. \end{aligned}$$

We note that  $\mathcal{F} = \{-a, -b\}$  is a founded repair and so, also a founded weak repair, for  $\langle \mathcal{I}, \eta \rangle$ . However,  $\mathcal{F}$  is not a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$  as  $\mathcal{F} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{F}) = \{-a, -b, -c\}$  is not a minimal set containing  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{F})$  and closed under  $\eta$ . One can check that  $\{-c\}$  also has these two properties.  $\square$

While stronger property than foundedness, being a justified weak repair still does not guarantee change-minimality (and so, the term *weak* cannot be dropped).

*Example 4.11.* Let us consider the set  $\eta$  of active integrity constraints

$$\begin{aligned} r_1 &= \text{not } a, b \supset +a \vee -b \\ r_2 &= a, \text{not } b \supset -a \vee +b \end{aligned}$$

and the set of update actions  $\mathcal{E} = \{+a, +b\}$ . It is easy to verify that  $\mathcal{E}$  is a justified weak repair of  $\mathcal{I} = \emptyset$ . Therefore, it ensures constraint enforcement and it is founded. However,  $\mathcal{E}$  is not minimal as  $\mathcal{I} = \emptyset$  is consistent with  $\eta$ , and the empty set is its only repair.  $\square$

Thus, to define justified repairs, as in the case of founded repairs, we need to impose change-minimality explicitly.

**Definition 4.12.** (JUSTIFIED REPAIR)

Let  $\mathcal{I}$  be a database and  $\eta$  a set of active integrity constraints. A set  $\mathcal{E}$  of update actions is a justified repair for  $\langle \mathcal{I}, \eta \rangle$  if  $\mathcal{E}$  is a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ , and for every  $\mathcal{E}' \subseteq \mathcal{E}$  such that  $\mathcal{I} \circ \mathcal{E}' \models \eta$ ,  $\mathcal{E}' = \mathcal{E}$ .  $\square$

Theorem 4.8 has yet another corollary, this time concerning justified and founded repairs.

**Corollary 4.13.** For each database  $\mathcal{I}$  and set of active integrity constraints  $\eta$ , if a set  $\mathcal{E}$  of update actions is a justified repair for  $\langle \mathcal{I}, \eta \rangle$  then  $\mathcal{E}$  is a founded repair for  $\langle \mathcal{I}, \eta \rangle$ .

**Proof:** Let  $\mathcal{E}$  be a justified repair for  $\langle \mathcal{I}, \eta \rangle$ . It follows by Theorem 4.8 that  $\mathcal{I} \circ \mathcal{E} \models \eta$ . Moreover, by the definition of justified repairs,  $\mathcal{E}$  is change minimal. Thus,  $\mathcal{E}$  is a repair. Again by Theorem 4.8,  $\mathcal{E}$  is founded. Thus,  $\mathcal{E}$  is a founded repair for  $\langle \mathcal{I}, \eta \rangle$ .  $\square$

Example 4.41 shows that the inclusion asserted by Corollary 4.13 is proper.

As illustrated by Example 4.11, in general, justified weak repairs form a proper subclass of justified repairs. However, in some cases the two concepts coincide. One such case is identified in the next theorem. The other important case is discussed in the next section.

**Theorem 4.14.** Let  $\mathcal{I}$  be a database and  $\eta$  a set of active integrity constraints such that for each update action  $\alpha \in \bigcup_{r \in \eta} \text{head}(r)$ ,  $\mathcal{I} \models \text{lit}(\alpha^D)$ . If  $\mathcal{E}$  is a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ , then  $\mathcal{E}$  is a justified repair for  $\langle \mathcal{I}, \eta \rangle$ .

**Proof:** Let  $\mathcal{E}$  be a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$  and let  $\mathcal{E}' \subseteq \mathcal{E}$  be such that  $\mathcal{I} \circ \mathcal{E}' \models \eta$ .

We define  $\mathcal{U} = \mathcal{E} \cup \text{ne}(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . By Theorem 4.7 and Proposition 4.5(2),  $\mathcal{U}$  is a minimal set of update actions containing  $\text{ne}(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  and closed under  $\eta$ . Let  $\mathcal{U}' = \mathcal{E}' \cup \text{ne}(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  and let  $r \in \eta$  be such that  $ua(\text{nup}(r)) \subseteq \mathcal{U}'$ . Since  $\mathcal{I} \circ \mathcal{E}' \models \eta$ ,  $\mathcal{I} \circ \mathcal{E}' \not\models \text{body}(r)$ . Thus, it follows that there is  $L \in \text{up}(r)$  such that  $\mathcal{I} \circ \mathcal{E}' \not\models L$ . Since  $L \in \text{up}(r)$ , there is  $\alpha \in \text{head}(r)$  such that  $L = \text{lit}(\alpha^D)$ . By the assumption,  $\mathcal{I} \models L$ , that is,  $\mathcal{I} \models \text{lit}(\alpha^D)$ . Since  $\mathcal{I} \circ \mathcal{E}' \not\models L$ ,  $\mathcal{I} \circ \mathcal{E}' \models \text{lit}(\alpha)$ . Thus,  $\alpha \in \mathcal{E}'$

and, consequently,  $\alpha \in \mathcal{U}'$ . It follows that  $\mathcal{U}'$  is closed under  $r$  and, since  $r$  was an arbitrary element of  $\eta$ , under  $\eta$ , too. Thus,  $\mathcal{U}' = \mathcal{U}$ , that is,  $\mathcal{E}' \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . Since  $\mathcal{E}' \subseteq \mathcal{E}$  and  $\mathcal{E} \cap ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \emptyset$ ,  $\mathcal{E}' = \mathcal{E}$ . It follows that  $\mathcal{E}$  is a minimal set of update actions such that  $\mathcal{I} \circ \mathcal{E} \models \eta$ .  $\square$

#### 4.4 Normal Active Integrity Constraints and Normalization

An active integrity constraint  $r$  is *normal* if  $|head(r)| = 1$ . We will now study properties of normal active integrity constraints. First, we will show that for that class of constraints, updating by justified weak repairs guarantees the minimality of change property and so, the explicit reference to the latter can be omitted from the definition of justified repairs.

**Theorem 4.15.** *Let  $\mathcal{I}$  be a database and  $\eta$  a set of normal active integrity constraints. If  $\mathcal{E}$  is a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$  then  $\mathcal{E}$  is a justified repair for  $\langle \mathcal{I}, \eta \rangle$ .*

**Proof:** Let  $\mathcal{E}$  be a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ . We have to prove that  $\mathcal{E}$  is minimal with respect to constraint enforcement. To this end, let us consider  $\mathcal{E}' \subseteq \mathcal{E}$  such that  $\mathcal{I} \circ \mathcal{E}' \models \eta$ .

We define  $\mathcal{U} = \mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  and  $\mathcal{U}' = \mathcal{E}' \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . We will show that  $\mathcal{U}'$  is closed under  $\eta$ . Let  $r \in \eta$  be such that  $ua(nup(r)) \subseteq \mathcal{U}'$ . Let  $\alpha$  be an update action such that  $head(r) = \{\alpha\}$ . Then  $body(r) = \{lit(\alpha^D)\} \cup nup(r)$ .

Since  $\mathcal{I} \circ \mathcal{E}' \models r$ ,  $\mathcal{I} \circ \mathcal{E}' \not\models body(r)$ . By our assumption,  $ua(nup(r)) \subseteq \mathcal{U}'$ . Thus,  $\mathcal{I} \circ \mathcal{U}' \models nup(r)$ . Since  $\mathcal{U}'$  is consistent, Proposition 4.5(2) implies that  $\mathcal{I} \circ \mathcal{E}' = \mathcal{I} \circ \mathcal{U}'$ . Thus,  $\mathcal{I} \circ \mathcal{E}' \not\models lit(\alpha^D)$  and, consequently,  $\mathcal{I} \circ \mathcal{E}' \models lit(\alpha)$ .

Since  $\mathcal{U}' \subseteq \mathcal{U}$ ,  $ua(nup(r)) \subseteq \mathcal{U}$ . By Theorem 4.7,  $\mathcal{U}$  is closed under  $\eta$ . Thus,  $\alpha \in \mathcal{U}$ . Since  $\mathcal{I} \circ \mathcal{U} = \mathcal{I} \circ \mathcal{E}$  (Proposition 4.5(2)),  $\mathcal{I} \circ \mathcal{E} \models lit(\alpha)$ .

If  $\mathcal{I} \models lit(\alpha)$  then, as  $\mathcal{I} \circ \mathcal{E} \models lit(\alpha)$ , we have  $\alpha \in ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) \subseteq \mathcal{U}'$ . If  $\mathcal{I} \not\models lit(\alpha)$  then, as  $\mathcal{I} \circ \mathcal{E}' \models lit(\alpha)$ , we have that  $\alpha \in \mathcal{E}' \subseteq \mathcal{U}'$ . Thus,  $\mathcal{U}'$  is closed under  $r$  and so, also under  $\eta$ . Consequently,  $\mathcal{U}' = \mathcal{U}$ . Since  $\mathcal{E} \cap ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \emptyset$ , it follows that  $\mathcal{E}' = \mathcal{E}$ . Thus,  $\mathcal{E}$  is a minimal set of update actions such that  $\mathcal{I} \circ \mathcal{E} \models \eta$ .  $\square$

Next, we introduce the operation of *normalization* of active integrity constraints, which consists of eliminating disjunctions from the heads of rules. For an active integrity constraint  $r = \phi \supset \alpha_1 \vee \dots \vee \alpha_n$ , by  $r^n$  we denote the set of *normal* active integrity constraints  $\{\phi \supset \alpha_1, \dots, \phi \supset \alpha_n\}$ . For a set  $\eta$  of active integrity constraints, we set  $\eta^n = \bigcup_{r \in \eta} r^n$ . It is shown in [29] that  $\mathcal{E}$  is founded for  $\langle \mathcal{I}, \eta \rangle$  if and only if  $\mathcal{E}$  is a founded (weak) repair for  $\langle \mathcal{I}, \eta^n \rangle$ . Thus,  $\mathcal{E}$  is a founded (weak) repair for  $\langle \mathcal{I}, \eta \rangle$  if and only if  $\mathcal{E}$  is a founded (weak) repair for  $\langle \mathcal{I}, \eta^n \rangle$ . For justified repairs, we have a weaker result. Normalization may eliminate some justified repairs.

**Theorem 4.16.** *Let  $\mathcal{I}$  be a database and  $\eta$  a set of active integrity constraints.*

1. *If a set  $\mathcal{E}$  of update actions is a justified repair for  $\langle \mathcal{I}, \eta^n \rangle$ , then  $\mathcal{E}$  is a justified repair for  $\langle \mathcal{I}, \eta \rangle$*

2. If a set  $\mathcal{E}$  of update actions is a justified weak repair for  $\langle \mathcal{I}, \eta^n \rangle$ , then  $\mathcal{E}$  is a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ .

**Proof:** Let  $\mathcal{E}$  be a justified repair for  $\langle \mathcal{I}, \eta^n \rangle$ . We define  $\mathcal{U} = \mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . By Corollary 4.13,  $\mathcal{E}$  is a founded repair for  $\langle \mathcal{I}, \eta^n \rangle$ . By a result from [29],  $\mathcal{E}$  is a founded repair for  $\langle \mathcal{I}, \eta \rangle$  and, consequently, a repair for  $\langle \mathcal{I}, \eta \rangle$ .

Since  $\mathcal{E}$  is, in particular, a justified weak repair for  $\langle \mathcal{I}, \eta^n \rangle$ ,  $\mathcal{U}$  is a justified action set for  $\langle \mathcal{I}, \eta^n \rangle$  (Theorem 4.7). Thus,  $\mathcal{U}$  is a minimal set of update actions containing  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  and closed under  $\eta^n$ . To prove that  $\mathcal{E}$  is a justified repair for  $\langle \mathcal{I}, \eta \rangle$ , it suffices to show that  $\mathcal{U}$  is a minimal set of update actions containing  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  and closed under  $\eta$ .

Let us consider an active integrity constraint

$$r = lit(\alpha_1^D), \dots, lit(\alpha_n^D), \phi \supset \alpha_1 \vee \dots \vee \alpha_n$$

in  $\eta$  such that  $ua(nup(r)) \subseteq \mathcal{U}$  (we note that  $nup(r)$  consists precisely of the literals that appear in  $\phi$ ). It follows that  $\mathcal{I} \circ \mathcal{U} \models nup(r)$ . Since  $\mathcal{E}$  is a repair,  $\mathcal{I} \circ \mathcal{E} \not\models body(r)$ . By Proposition 4.5(2),  $\mathcal{I} \circ \mathcal{E} = \mathcal{I} \circ \mathcal{U}$ . Thus,  $\mathcal{I} \circ \mathcal{U} \not\models body(r)$ . It follows that there is  $i$ ,  $1 \leq i \leq n$ , such that  $\mathcal{I} \circ \mathcal{U} \not\models lit(\alpha_i^D)$ . Thus,  $\alpha_i^D \notin \mathcal{U}$ . By Theorem 4.8(1),  $\alpha_i \in \mathcal{U}$ . Thus,  $\mathcal{U}$  is closed under  $r$  and, consequently, under  $\eta$ , as well.

We will now show that  $\mathcal{U}$  is minimal in the class of sets of update actions containing  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  and closed under  $\eta$ . Let  $\mathcal{U}'$  be a set of update actions such that  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) \subseteq \mathcal{U}' \subseteq \mathcal{U}$  and  $\mathcal{U}'$  is closed under  $\eta$ . Let us consider an active integrity constraint in  $s \in \eta^n$  such that  $ua(nup(s)) \subseteq \mathcal{U}'$ .

By the definition of  $\eta^n$ , there is an active integrity constraint  $r \in \eta$  such that

$$r = lit(\alpha_1^D), \dots, lit(\alpha_i^D), \dots, lit(\alpha_n^D), \phi \supset \alpha_1 \vee \dots \vee \alpha_i \vee \dots \vee \alpha_n$$

and

$$s = lit(\alpha_1^D), \dots, lit(\alpha_i^D), \dots, lit(\alpha_n^D), \phi \supset \alpha_i.$$

Since  $ua(nup(s)) \subseteq \mathcal{U}'$ ,  $ua(nup(r)) \subseteq \mathcal{U}'$ . As  $\mathcal{U}'$  is closed under  $\eta$ , there is  $j$ ,  $1 \leq j \leq n$ , such that  $\alpha_j \in \mathcal{U}'$ . For every  $k$  such that  $1 \leq k \leq n$  and  $k \neq i$ ,  $\alpha_k^D \in \mathcal{U}'$ . By the consistency of  $\mathcal{U}'$ , we conclude that  $\alpha_i \in \mathcal{U}'$ . Thus,  $\mathcal{U}'$  is closed under  $s$  and, consequently, under  $\eta^n$ . Since  $\mathcal{U}' \subseteq \mathcal{U}$  and  $\mathcal{U}$  is minimal containing  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  and closed under  $\eta^n$  it follows that  $\mathcal{U}' = \mathcal{U}$ . Thus,  $\mathcal{U}$  is minimal containing  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  and closed under  $\eta$ . Consequently,  $\mathcal{E}$  is a justified repair for  $\langle \mathcal{I}, \eta \rangle$ .

(2) If  $\mathcal{E}$  is a justified weak repair for  $\langle \mathcal{I}, \eta^n \rangle$  then, by Theorem 4.15,  $\mathcal{E}$  is a justified repair for  $\langle \mathcal{I}, \eta^n \rangle$ . By (1),  $\mathcal{E}$  is a justified repair for  $\langle \mathcal{I}, \eta \rangle$  and so, a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ .  $\square$

The following example shows that the inclusions in the previous theorem are, in general, proper.

*Example 4.17.* Let us consider an empty database  $\mathcal{I}$ , the set  $\eta$  of active integrity constraints

$$\begin{aligned}
r_1 &= \text{not } a, \text{not } b \supset +a \vee +b \\
r_2 &= a, \text{not } b \supset +b \\
r_3 &= \text{not } a, b \supset +a
\end{aligned}$$

its normalized version  $\eta^n$

$$\begin{aligned}
r_{1,1} &= \text{not } a, \text{not } b \supset +a \\
r_{1,2} &= \text{not } a, \text{not } b \supset +b \\
r_{2,1} &= a, \text{not } b \supset +b \\
r_{3,1} &= \text{not } a, b \supset +a
\end{aligned}$$

and the set of update actions  $\mathcal{E} = \{+a, +b\}$ . It is easy to verify that  $\mathcal{E}$  is a justified repair for  $\langle \mathcal{I}, \eta \rangle$ . However,  $\mathcal{E}$  is not a justified weak repair for  $\langle \mathcal{I}, \eta^n \rangle$  (and so a justified repair for  $\langle \mathcal{I}, \eta^n \rangle$ ). Indeed, it is not a minimal set containing  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \emptyset$  and closed under  $\eta^n$  as  $\emptyset$  is also closed under  $\eta^n$ .  $\square$

## 4.5 Shifting Theorem

In this section we study the concept of shifting [108]. Shifting consists of transforming an instance  $\langle \mathcal{I}, \eta \rangle$  of the database repair problem to a syntactically isomorphic instance  $\langle \mathcal{I}', \eta' \rangle$  by changing integrity constraints to reflect the “shift” of  $\mathcal{I}$  into  $\mathcal{I}'$ . A semantics for database repair problem has the *shifting property* if the repairs of the “shifted” instance of the database update problem are precisely the results of modifying the repairs of the original instance according to the shift from  $\mathcal{I}$  to  $\mathcal{I}'$ . The shifting property is important. If a semantics of database updates has it, the study of that semantics can be reduced to the case when the input database is the empty set. In many cases it allows us to relate a semantics of database repairs to some semantics of logic programs with negation.

*Example 4.18.* Let  $\mathcal{I} = \{a, b\}$  and let  $\eta = \{a, b \supset -a \vee -b\}$ . There are two founded repairs for  $\langle \mathcal{I}, \eta \rangle$ :  $\mathcal{E}_1 = \{-a\}$  and  $\mathcal{E}_2 = \{-b\}$ . Let  $\mathcal{W} = \{a\}$ . We will now “shift” the instance  $\langle \mathcal{I}, \eta \rangle$  with respect to  $\mathcal{W}$ . To this end, we will first modify  $\mathcal{I}$  by changing the status in  $\mathcal{I}$  of elements in  $\mathcal{W}$ , in our case, of  $a$ . Since  $a \in \mathcal{I}$ , we will remove it. Thus,  $\mathcal{I}$  “shifted” with respect to  $\mathcal{W}$  becomes  $\mathcal{I}' = \{b\}$ . Next, we will modify  $\eta$  correspondingly, replacing literals and update actions involving  $a$  by their duals. That results in  $\eta' = \{\text{not } a, b \supset +a \vee -b\}$ . One can check that the resulting instance  $\langle \mathcal{I}', \eta' \rangle$  of the update problem has two founded repairs:  $\{+a\}$  and  $\{-b\}$ . Moreover, they can be obtained from the founded repairs for  $\langle \mathcal{I}, \eta \rangle$  by consistently replacing  $-a$  with  $+a$  and  $+a$  with  $-a$  (the latter does not apply in this example).  $\square$

The situation presented in Example 4.18 is not coincidental. In this section we will show that the semantics of (weak) repairs, founded (weak) repairs and justified (weak) repairs satisfy the shifting property.

We start by observing that *shifting* a database  $\mathcal{I}$  to a database  $\mathcal{I}'$  can be modeled by means of the symmetric difference operator. Namely, we have  $\mathcal{I}' = \mathcal{I} \div \mathcal{W}$ , where



$\mathcal{W} = \mathcal{I} \div i'$ . This identity shows that one can shift any database  $\mathcal{I}$  into any database  $\mathcal{I}'$  by forming a symmetric difference of  $\mathcal{I}$  with some set of atom  $\mathcal{W}$  (specifically,  $\mathcal{W} = \mathcal{I} \div i'$ ). We will now extend the operation of shifting a database with respect to  $\mathcal{W}$  to the case of literals, update actions and integrity constraints. To this end, we introduce a *shifting* operator  $T_{\mathcal{W}}$ .

**Definition 4.19.** *Let  $\mathcal{W}$  be a database and  $\ell$  a literal or an update action. We define*

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

and we extend this definition to sets of literals or update actions, respectively.

Furthermore, if  $op$  is an operator on sets of literals or update actions (such as conjunction or disjunction), for every set  $X$  of literals or update actions, we define

$$T_{\mathcal{W}}(op(X)) = op(T_{\mathcal{W}}(X)).$$

Finally, for an active integrity constraint  $r = \phi \supset \psi$ , we set

$$T_{\mathcal{W}}(r) = T_{\mathcal{W}}(\phi) \supset T_{\mathcal{W}}(\psi)$$

and we extend the notation to sets active integrity constraints in the standard way.

□

To illustrate the last two parts of the definition, we note that when  $op$  stands for the conjunction of a set of literals and  $X = \{L_1, \dots, L_n\}$ , where every  $L_i$  is a literal,  $T_{\mathcal{W}}(op(X)) = op(T_{\mathcal{W}}(X))$  specializes to

$$T_{\mathcal{W}}(L_1, \dots, L_n) = T_{\mathcal{W}}(L_1), \dots, T_{\mathcal{W}}(L_n).$$

Similarly, for an active integrity constraint

$$r = L_1, \dots, L_n \supset \alpha_1 \vee \dots \vee \alpha_m$$

we obtain

$$T_{\mathcal{W}}(r) = T_{\mathcal{W}}(L_1), \dots, T_{\mathcal{W}}(L_n) \supset T_{\mathcal{W}}(\alpha_1) \vee \dots \vee T_{\mathcal{W}}(\alpha_m).$$

Clearly, we overload the notation  $T_{\mathcal{W}}$  and interpret it based on the type of the argument. We will now present several useful properties of the operator  $T_{\mathcal{W}}$ .

**Proposition 4.20.** *Let  $\mathcal{W}$  be a database.*

1. For every update action  $\alpha$ ,  $T_{\mathcal{W}}(\text{lit}(\alpha)) = \text{lit}(T_{\mathcal{W}}(\alpha))$
2. For every set  $A$  of literals (update actions, active integrity constraints, respectively)  $T_{\mathcal{W}}(T_{\mathcal{W}}(A)) = A$
3. For every consistent set  $\mathcal{A}$  of literals (update actions, respectively),  $T_{\mathcal{W}}(\mathcal{A})$  is consistent

4. For every databases  $\mathcal{I}$  and  $\mathcal{R}$ ,  $T_{\mathcal{W}}(ne(\mathcal{I}, \mathcal{R})) = ne(\mathcal{I} \div \mathcal{W}, \mathcal{R} \div \mathcal{W})$   
 5. For every active integrity constraint  $r$ ,  $nup(T_{\mathcal{W}}(r)) = T_{\mathcal{W}}(nup(r))$ .

**Proof:** (1) - (3) follow directly from the definitions. We omit the details.

(4) Let  $\alpha \in ne(\mathcal{I} \div \mathcal{W}, \mathcal{R} \div \mathcal{W})$ . If  $\alpha = +a$ , then it follows that  $a \in (\mathcal{I} \div \mathcal{W}) \cap (\mathcal{R} \div \mathcal{W})$ . Let us assume that  $a \in \mathcal{W}$ . Then  $a \notin i \cup \mathcal{R}$  and, consequently,  $-a \in ne(\mathcal{I}, \mathcal{R})$ . Since  $a \in \mathcal{W}$ ,  $+a = T_{\mathcal{W}}(-a)$ . Thus,  $\alpha \in T_{\mathcal{W}}(ne(\mathcal{I}, \mathcal{R}))$ . The case when  $\alpha = -a$  can be dealt with in a similar way. It follows that  $ne(\mathcal{I} \div \mathcal{W}, \mathcal{R} \div \mathcal{W}) \subseteq T_{\mathcal{W}}(ne(\mathcal{I}, \mathcal{R}))$ .

Let  $\mathcal{I}' = \mathcal{I} \div \mathcal{W}$  and  $\mathcal{R}' = \mathcal{R} \div \mathcal{W}$ . Then  $\mathcal{I} = \mathcal{I}' \div \mathcal{W}$ ,  $\mathcal{R} = \mathcal{R}' \div \mathcal{W}$  and, by applying the inclusion we just proved to  $\mathcal{I}'$  and  $\mathcal{R}'$ , we obtain

$$ne(\mathcal{I}, \mathcal{R}) = ne(\mathcal{I}' \div \mathcal{W}, \mathcal{R}' \div \mathcal{W}) \subseteq T_{\mathcal{W}}(ne(\mathcal{I}', \mathcal{R}')).$$

Consequently,

$$T_{\mathcal{W}}(ne(\mathcal{I}, \mathcal{R})) \subseteq T_{\mathcal{W}}(T_{\mathcal{W}}(ne(\mathcal{I}', \mathcal{R}'))) = ne(\mathcal{I} \div \mathcal{W}, \mathcal{R} \div \mathcal{W}).$$

Thus, the claim follows.

(5) Let  $L \in nup(T_{\mathcal{W}}(r))$ . It follows that  $L \in body(T_{\mathcal{W}}(r))$  and  $L^D \notin lit(head(T_{\mathcal{W}}(r)))$ . Clearly,  $head(T_{\mathcal{W}}(r)) = T_{\mathcal{W}}(head(r))$  and  $body(T_{\mathcal{W}}(r)) = T_{\mathcal{W}}(body(r))$ . Thus,  $L \in T_{\mathcal{W}}(body(r))$  and  $L^D \notin T_{\mathcal{W}}(head(r))$ . Consequently,  $T_{\mathcal{W}}(L) \in body(r)$ . Moreover, since  $T_{\mathcal{W}}(L^D) = (T_{\mathcal{W}}(L))^D$ ,  $(T_{\mathcal{W}}(L))^D \notin head(r)$ . It follows that  $T_{\mathcal{W}}(L) \in nup(r)$  and so,  $L \in T_{\mathcal{W}}(nup(r))$ . Hence,  $nup(T_{\mathcal{W}}(r)) \subseteq T_{\mathcal{W}}(nup(r))$ .

Applying this inclusion to an active integrity constraint  $s = T_{\mathcal{W}}(r)$ , we obtain  $nup(r) \subseteq T_{\mathcal{W}}(nup(T_{\mathcal{W}}(r)))$ , which implies  $T_{\mathcal{W}}(nup(r)) \subseteq T_{\mathcal{W}}(T_{\mathcal{W}}(nup(T_{\mathcal{W}}(r)))) = nup(T_{\mathcal{W}}(r))$ . Thus, the equality  $nup(T_{\mathcal{W}}(r)) = T_{\mathcal{W}}(nup(r))$  follows.  $\square$

**Proposition 4.21.** *Let  $\mathcal{I}$  and  $\mathcal{W}$  be databases and let  $L$  be a literal or an update action. Then  $\mathcal{I} \models L$  if and only if  $\mathcal{I} \div \mathcal{W} \models T_{\mathcal{W}}(L)$ .*

**Proof:** ( $\Rightarrow$ ) Let us assume that  $\mathcal{I} \models L$ . If  $L = a$ , where  $a$  is an atom, then  $a \in \mathcal{I}$ . There are two cases:  $a \in \mathcal{W}$  and  $a \notin \mathcal{W}$ . In the first case,  $a \notin \mathcal{I} \div \mathcal{W}$  and  $T_{\mathcal{W}}(a) = not\ a$ . In the second case,  $a \in i \div \mathcal{W}$  and  $T_{\mathcal{W}}(a) = a$ . In each case,  $\mathcal{I} \div \mathcal{W} \models T_{\mathcal{W}}(a)$ , that is,  $\mathcal{I} \div \mathcal{W} \models T_{\mathcal{W}}(L)$ .

The case  $L = not\ a$ , where  $a$  is an atom, is similar. First, we have that  $a \notin i$ . If  $a \in \mathcal{W}$  then  $a \in i \div \mathcal{W}$  and  $T_{\mathcal{W}}(not\ a) = a$ . If  $a \notin \mathcal{W}$  then  $a \notin i \div \mathcal{W}$  and  $T_{\mathcal{W}}(not\ a) = not\ a$ . In each case,  $\mathcal{I} \div \mathcal{W} \models T_{\mathcal{W}}(not\ a)$ , that is,  $\mathcal{I} \div \mathcal{W} \models T_{\mathcal{W}}(L)$ .

( $\Leftarrow$ ) Let us assume that  $\mathcal{I} \div \mathcal{W} \models T_{\mathcal{W}}(L)$ . Then,  $(\mathcal{I} \div \mathcal{W}) \div \mathcal{W} = \mathcal{I}$  and  $T_{\mathcal{W}}(T_{\mathcal{W}}(L)) = L$ . Thus,  $\mathcal{I} \models L$  follows by the implication ( $\Rightarrow$ ).  $\square$

**Proposition 4.22.** *Let  $\mathcal{I}$  and  $\mathcal{W}$  be databases, and let  $\mathcal{U}$  be a consistent set of update actions. Then  $(\mathcal{I} \circ \mathcal{U}) \div \mathcal{W} = (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U})$ .*

**Proof:** We note that since  $\mathcal{U}$  is consistent,  $T_{\mathcal{W}}(\mathcal{U})$  is consistent, too. Thus, *both* sides of the identity are well defined.

Let  $a \in (\mathcal{I} \circ \mathcal{U}) \div \mathcal{W}$ . If  $+a \in T_{\mathcal{W}}(\mathcal{U})$ , then  $a \in (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U})$ . Thus, let us assume that  $+a \notin T_{\mathcal{W}}(\mathcal{U})$ . We have two cases.

Case 1:  $a \notin \mathcal{W}$ . From the definition of  $T_{\mathcal{W}}$ ,  $+a \notin \mathcal{U}$ . Since  $a \in (\mathcal{I} \circ \mathcal{U}) \div \mathcal{W}$ ,  $a \in i \circ \mathcal{U}$  and, consequently,  $a \in i$  and  $-a \notin \mathcal{U}$ . Thus,  $a \in (\mathcal{I} \div \mathcal{W})$  and  $-a \notin T_{\mathcal{W}}(\mathcal{U})$  (otherwise, as  $T_{\mathcal{W}}(-a) = -a$ , we would have  $-a \in \mathcal{U}$ ). Consequently,  $a \in (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U})$ .

Case 2:  $a \in \mathcal{W}$ . From the definition of  $T_{\mathcal{W}}$ ,  $-a \notin \mathcal{U}$ . Since  $a \in (\mathcal{I} \circ \mathcal{U}) \div \mathcal{W}$ ,  $a \notin i \circ \mathcal{U}$ . Thus,  $a \notin i$  and  $+a \notin \mathcal{U}$ . It follows that  $a \in i \div \mathcal{W}$  and  $-a \notin T_{\mathcal{W}}(\mathcal{U})$  (otherwise we would have  $+a \in \mathcal{U}$ , as  $T_{\mathcal{W}}(-a) = +a$ , in this case). Hence,  $a \in (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U})$ .

If  $a \notin (\mathcal{I} \circ \mathcal{U}) \div \mathcal{W}$ , we reason similarly. If  $-a \in T_{\mathcal{W}}(\mathcal{U})$ , then  $a \notin (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U})$ . Therefore, let us assume that  $-a \notin T_{\mathcal{W}}(\mathcal{U})$ . As before, there are two cases.

Case 1:  $a \notin \mathcal{W}$  and thus  $-a \notin \mathcal{U}$ . Since  $a \notin (\mathcal{I} \circ \mathcal{U}) \div \mathcal{W}$ ,  $a \notin i \circ \mathcal{U}$  and, consequently,  $a \notin i$  and  $+a \notin \mathcal{U}$ . Thus,  $a \notin (\mathcal{I} \div \mathcal{W})$  and  $+a \notin T_{\mathcal{W}}(\mathcal{U})$ . Consequently,  $a \notin (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U})$ .

Case 2:  $a \in \mathcal{W}$  and thus  $+a \notin \mathcal{U}$ . In this case,  $a \in \mathcal{I} \circ \mathcal{U}$ . Thus,  $a \in i$  and  $-a \notin \mathcal{U}$ . It follows that  $a \notin i \div \mathcal{W}$  and  $+a \notin T_{\mathcal{W}}(\mathcal{U})$ . Hence,  $a \notin (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U})$ .  $\square$

**Corollary 4.23.** *Let  $\mathcal{I}$  and  $\mathcal{W}$  be databases,  $\mathcal{U}$  a consistent set of update actions, and  $L$  a literal or an action update. Then  $\mathcal{I} \circ \mathcal{U} \models L$  if and only if  $(\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U}) \models T_{\mathcal{W}}(L)$*

**Proof:** By Proposition 4.21,  $\mathcal{I} \circ \mathcal{U} \models L$  if and only if  $(\mathcal{I} \circ \mathcal{U}) \div \mathcal{W} \models T_{\mathcal{W}}(L)$ . By Proposition 4.22, the latter condition is equivalent to the condition  $(\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U}) \models T_{\mathcal{W}}(L)$ .  $\square$

**Theorem 4.24.** (SHIFTING THEOREM FOR (WEAK) REPAIRS AND FOUNDED (WEAK) REPAIRS)

*Let  $\mathcal{I}$  and  $\mathcal{W}$  be databases. For every set  $\eta$  of active integrity constraints and for every consistent set  $\mathcal{E}$  of update actions, we have*

1.  $\mathcal{E}$  is a weak repair for  $\langle \mathcal{I}, \eta \rangle$  if and only if  $T_{\mathcal{W}}(\mathcal{E})$  is a weak repair for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$
2.  $\mathcal{E}$  is a repair for  $\langle \mathcal{I}, \eta \rangle$  if and only if  $T_{\mathcal{W}}(\mathcal{E})$  is a repair for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$
3.  $\mathcal{E}$  is founded for  $\langle \mathcal{I}, \eta \rangle$  if and only if  $T_{\mathcal{W}}(\mathcal{E})$  is founded for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ .
4.  $\mathcal{E}$  is a founded (weak) repair for  $\langle \mathcal{I}, \eta \rangle$  if and only if  $T_{\mathcal{W}}(\mathcal{E})$  is a founded (weak) repair for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ .

**Proof.** (1) Let us assume that  $\mathcal{E}$  is a weak repair for  $\langle \mathcal{I}, \eta \rangle$ . It follows that  $\mathcal{E}$  is consistent. Since  $\mathcal{I} \circ \mathcal{E} \models \eta$ , by Corollary 4.23,  $(\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{E}) \models T_{\mathcal{W}}(\eta)$ . The converse implication follows from the one we just proved by Proposition 4.20(2).

(2) As before, it suffices to show only one implication. Let  $\mathcal{E}$  be a repair for  $\langle \mathcal{I}, \eta \rangle$ . Then,  $\mathcal{E}$  is a weak repair for  $\langle \mathcal{I}, \eta \rangle$ . By (1),  $\mathcal{E}$  is a weak repair for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ .

Let  $\mathcal{E}' \subseteq T_{\mathcal{W}}(\mathcal{E})$  be such that  $(\mathcal{I} \div \mathcal{W}) \circ \mathcal{E}' \models T_{\mathcal{W}}(\eta)$ . It follows that  $T_{\mathcal{W}}(\mathcal{E}') \subseteq T_{\mathcal{W}}(T_{\mathcal{W}}(\mathcal{E})) = \mathcal{E}$ . Since  $\mathcal{E}$  is consistent,  $T_{\mathcal{W}}(\mathcal{E}')$  is consistent, too. By Corollary 4.23 and Proposition 4.20(2), since  $(\mathcal{I} \div \mathcal{W}) \circ \mathcal{E}' \models T_{\mathcal{W}}(\eta)$ , then  $\mathcal{I} \circ T_{\mathcal{W}}(\mathcal{E}') \models \eta$ . Since  $\mathcal{E}$  is a repair and  $T_{\mathcal{W}}(\mathcal{E}') \subseteq \mathcal{E}$ ,  $T_{\mathcal{W}}(\mathcal{E}') = \mathcal{E}$ . Thus,  $\mathcal{E}' = T_{\mathcal{W}}(\mathcal{E})$  and so,  $T_{\mathcal{W}}(\mathcal{E})$  is a repair for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ .

(3) As in two previous cases, we show only one implication. Thus, let us assume that  $\mathcal{E}$  is founded for  $\langle \mathcal{I}, \eta \rangle$ . Let  $\alpha \in T_{\mathcal{W}}(\mathcal{E})$ . It follows that there is  $\beta \in \mathcal{E}$  such that  $\alpha = T_{\mathcal{W}}(\beta)$ . Since  $\mathcal{E}$  is founded with respect to  $\langle \mathcal{I}, \eta \rangle$ , there is an active integrity constraint  $r$  such that  $\beta \in \text{head}(r)$ ,  $\mathcal{I} \circ \mathcal{E} \models \text{nup}(r)$ , and for every  $\gamma \in \text{head}(r) \setminus \{\beta\}$ ,  $\mathcal{I} \circ \mathcal{E} \models \gamma^D$ .

Clearly, the active integrity constraint  $T_{\mathcal{W}}(r)$  belongs to  $T_{\mathcal{W}}(\eta)$  and  $\alpha = T_{\mathcal{W}}(\beta)$  is an element of  $\text{head}(T_{\mathcal{W}}(r))$ . By Proposition 4.20(5),  $\text{nup}(r) = \text{nup}(T_{\mathcal{W}}(r))$ . Thus, by Corollary 4.23,  $(\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{E}) \models \text{nup}(T_{\mathcal{W}}(r))$ . Next, let  $\gamma \in \text{head}(T_{\mathcal{W}}(r)) \setminus \{\alpha\}$ . Then, there is  $\delta \in \text{head}(r) \setminus \{\beta\}$  such that  $\gamma = T_{\mathcal{W}}(\delta)$ . Since  $\mathcal{I} \circ \mathcal{E} \models \gamma^D$ , it follows that  $(\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{E}) \models T_{\mathcal{W}}(\delta^D)$ , that is,  $(\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{E}) \models \gamma^D$ . Thus,  $\alpha$  is founded with respect to  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$  and  $T_{\mathcal{W}}(\mathcal{E})$  and  $T_{\mathcal{W}}(\mathcal{E})$  is founded with respect to  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ .

(4) This property is a direct consequence of (1), (2), and (3).  $\square$

We will now turn our attention to justified repairs. We need one more auxiliary result.

**Lemma 4.25.** *Let  $\mathcal{I}$  and  $\mathcal{W}$  be databases. For every set  $\eta$  of active integrity constraints and for every set  $\mathcal{U}$  of update actions,  $\mathcal{U}$  is a justified action set for  $\langle \mathcal{I}, \eta \rangle$  if and only if  $T_{\mathcal{W}}(\mathcal{U})$  is a justified action set for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ .*

**Proof:** ( $\Rightarrow$ ) We have to prove that  $T_{\mathcal{W}}(\mathcal{U})$  is consistent, and minimal among all supersets of  $ne(\mathcal{I} \div \mathcal{W}, (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U}))$  that are closed under  $T_{\mathcal{W}}(\eta)$ .

Since  $\mathcal{U}$  is a justified action set for  $\langle \mathcal{I}, \eta \rangle$ ,  $\mathcal{U}$  is consistent and  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U}) \subseteq \mathcal{U}$ . The former implies that  $T_{\mathcal{W}}(\mathcal{U})$  is consistent (cf. Proposition 4.20(1)). The latter implies that  $ne(\mathcal{I} \div \mathcal{W}, (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U})) \subseteq T_{\mathcal{W}}(\mathcal{U})$  (cf. Propositions 4.20(2) and 4.22).

Next, we prove that  $T_{\mathcal{W}}(\mathcal{U})$  is closed under  $T_{\mathcal{W}}(\eta)$ . Let  $r$  be an active integrity constraint in  $T_{\mathcal{W}}(\eta)$  such that  $\text{body}(r)$  is consistent,  $\text{nup}(r) \subseteq \text{lit}(T_{\mathcal{W}}(\mathcal{U}))$ . Then, there exists  $s \in \eta$  such that  $r = T_{\mathcal{W}}(s)$ . By Proposition 4.20(5),  $\text{nup}(r) = T_{\mathcal{W}}(\text{nup}(s))$ . As  $T_{\mathcal{W}}(\text{nup}(s)) \subseteq \text{lit}(T_{\mathcal{W}}(\mathcal{U}))$ , we have that  $\text{nup}(s) \subseteq \text{lit}(\mathcal{U})$ . Since  $\mathcal{U}$  is closed under  $s$ , there exists  $\alpha \in \text{head}(s)$  such that  $\alpha \in \mathcal{U}$ . Thus, we obtain that  $T_{\mathcal{W}}(\alpha) \in T_{\mathcal{W}}(\text{head}(s)) = \text{head}(r)$ , and that  $T_{\mathcal{W}}(\alpha) \in T_{\mathcal{W}}(\mathcal{U})$ . Consequently,  $\text{head}(r) \cap T_{\mathcal{W}}(\mathcal{U}) \neq \emptyset$ . It follows that  $T_{\mathcal{W}}(\mathcal{U})$  is closed under  $r$  and so, also under  $T_{\mathcal{W}}(\eta)$ .

Finally, let us consider a set  $\mathcal{V}$  of update actions such that  $ne(\mathcal{I} \div \mathcal{W}, (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U})) \subseteq \mathcal{V} \subseteq T_{\mathcal{W}}(\mathcal{U})$  and closed under  $T_{\mathcal{W}}(\eta)$ . By Propositions 4.20(2) and 4.22,  $ne(\mathcal{I} \div \mathcal{W}, (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{U})) = T_{\mathcal{W}}(ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U}))$ . Thus,  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U}) \subseteq T_{\mathcal{W}}(\mathcal{V}) \subseteq \mathcal{U}$ . From the fact that  $\mathcal{V}$  is closed under  $T_{\mathcal{W}}(\eta)$  it follows that  $T_{\mathcal{W}}(\mathcal{V})$  is closed under  $\eta$  (one can show it reasoning similarly as in the previous paragraph). As  $\mathcal{U}$  is minimal in the class of supersets of  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$  closed under  $\eta$ ,  $T_{\mathcal{W}}(\mathcal{V}) = \mathcal{U}$  and so,  $\mathcal{V} = T_{\mathcal{W}}(\mathcal{U})$ . This completes the proof of the implication ( $\Rightarrow$ ).

( $\Leftarrow$ ) If  $T_{\mathcal{W}}(\mathcal{U})$  is a justified action set for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ , the implication ( $\Rightarrow$ ) yields that  $T_{\mathcal{W}}(T_{\mathcal{W}}(\mathcal{U})) = \mathcal{U}$  is a justified action set for  $\langle (\mathcal{I} \div \mathcal{W}) \div \mathcal{W} = \mathcal{I}, \eta \rangle$ .  $\square$

This result implies the shifting property for the semantics of justified revisions.

**Theorem 4.26.** (SHIFTING THEOREM FOR (WEAK) JUSTIFIED REPAIRS) *Let  $\mathcal{I}$  and  $\mathcal{W}$  be databases. For every set  $\eta$  of active integrity constraints and for every set  $\mathcal{E}$  of update actions,  $\mathcal{E}$  is an justified (weak) repair for  $\langle \mathcal{I}, \eta \rangle$  if and only if  $T_{\mathcal{W}}(\mathcal{E})$  is a justified (weak) repair for  $\langle \mathcal{I}, T_{\mathcal{W}}(\eta) \rangle$ .*

**Proof:** ( $\Rightarrow$ ) If  $\mathcal{E}$  is a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$ , then  $\mathcal{E} \cap ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \emptyset$  and  $\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$  is a justified action set for  $\langle \mathcal{I}, \eta \rangle$  (Theorem 4.7). It follows that  $T_{\mathcal{W}}(\mathcal{E}) \cap T_{\mathcal{W}}(ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})) = \emptyset$ . Moreover, by Lemma 4.25,  $T_{\mathcal{W}}(\mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}))$  is a justified action set for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ .

We have  $T_{\mathcal{W}}(ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})) = ne(\mathcal{I} \div \mathcal{W}, (\mathcal{I} \div \mathcal{W}) \circ T_{\mathcal{W}}(\mathcal{E}))$ . Thus, again by Theorem 4.7,  $T_{\mathcal{W}}(\mathcal{E})$  is a justified weak repair for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ .

If  $\mathcal{E}$  is a justified repair for  $\langle \mathcal{I}, \eta \rangle$ , then our argument shows that  $T_{\mathcal{W}}(\mathcal{E})$  is a justified weak repair for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ . Moreover, since  $\mathcal{E}$  is a repair for  $\mathcal{I}$ , by Theorem 4.24(2) we have that  $T_{\mathcal{W}}(\mathcal{E})$  is a repair for  $\mathcal{I} \div \mathcal{W}$ . It follows that  $T_{\mathcal{W}}(\mathcal{E})$  is a justified repair for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\eta) \rangle$ .

( $\Leftarrow$ ) This implication follows from the other one in the same way as in several other similar cases in the chapter.  $\square$

Theorems 4.24 and 4.26 imply that in the context of (weak) repairs, founded (weak) repairs or justified (weak) repairs, an instance  $\langle \mathcal{I}, \eta \rangle$  of the database update problem can be shifted to the instance the empty initial database. That property simplifies studies of these semantics as it allows us to eliminate one parameter (the initial database) from considerations.

**Corollary 4.27.** *Let  $\mathcal{I}$  be a database and  $\eta$  a set of active integrity constraints. Then  $\mathcal{E}$  is a weak repair (repair, founded weak repair, founded repair, justified weak repair, ustified repair, respectively) for  $\langle \mathcal{I}, \eta \rangle$  if and only if  $T_{\mathcal{I}}(\mathcal{E})$  is a weak repair (repair, founded weak repair, founded repair, justified weak repair, ustified repair, respectively) for  $\langle \emptyset, T_{\mathcal{I}}(\eta) \rangle$ .  $\square$*

## 4.6 Complexity and Computation

We noted earlier that the problem of the existence of a (weak) repair is NP-complete, and the same is true for the problem of the existence of founded weak repairs. On the other hand, the problem of the existence of a founded repair is  $\Sigma_P^2$ -complete [29]. In this section, we study the problem of the existence of justified (weak) repairs.

For our hardness results, we will use problems in logic programming. We will consider disjunctive and normal logic programs that satisfy some additional syntactic constraints. Namely, we will consider only programs without rules which contain

multiple occurrences of the same atom (that is, in the head and in the body, negated or not; or in the body — both positively and negatively). We call such programs *simple*. It is well known that the problem of the existence of a stable model of a normal logic program is NP-complete [107], and of the disjunctive logic program —  $\Sigma_2^P$ -complete [51]. The proofs in [51, 107] imply that the results hold also under the restriction to simple normal and simple disjunctive programs, respectively (in the case of disjunctive logic programs, a minor modification of the construction is required).

Let  $\rho$  be a logic programming rule, say

$$\rho = a_1 \vee \dots \vee a_k \leftarrow \beta.$$

We define

$$aic(\rho) = \text{not } a_1, \dots, \text{not } a_k, \beta \supset +a_1 \vee \dots \vee +a_k.$$

We extend the operator  $aic(\cdot)$  to logic programs in a standard way. We note that if a rule  $\rho$  is simple then  $body(aic(\rho))$  is consistent and  $nup(aic(\rho)) = body(\rho)$ .

We recall that a set  $M$  of atoms is an answer set of a disjunctive logic program  $P$  if  $M$  is a minimal set closed under the reduct  $P^M$ , where  $P^M$  consists of the rules obtained by dropping all negative literals from those rules in  $P$  that do not contain a literal  $\text{not } a$  in the body, for any  $a \in M$  (we refer to [66] for details). Our first two lemmas establish a result needed for hardness arguments.

**Lemma 4.28.** *Let  $P$  be a simple disjunctive logic program and  $M', M$  sets of atoms such that  $M' \subseteq M$ . Then  $M'$  is a model of  $P^M$  if and only if  $\{+a \mid a \in M'\} \cup \{-a \mid a \notin M'\}$  is closed under  $aic(P)$ .*

**Proof:** Let us define  $\mathcal{U} = \{+a \mid a \in M'\} \cup \{-a \mid a \notin M'\}$ . We note that  $\mathcal{U}$  is consistent.

( $\Rightarrow$ ) Let  $r \in aic(P)$ ,  $\rho \in P$  be a rule such that  $r = aic(\rho)$ , and  $\rho'$  be the rule obtained by eliminating from  $\rho$  all negative literals.

Since  $P$  is simple,  $nup(r) = body(\rho)$ . Let us assume that  $nup(r) \subseteq \mathcal{U}$ . It follows that  $\rho' \in P^M$  and that  $M' \models body(\rho')$ . Thus,  $head(\rho') \cap M' \neq \emptyset$ . Since  $head(\rho) = head(\rho')$  and  $head(r) = head(aic(\rho)) = ua(head(\rho))$ ,  $head(r) \cap \mathcal{U} \neq \emptyset$ . That is,  $\mathcal{U}$  is closed under  $r$  and, since  $r$  was chosen arbitrarily, under  $aic(P)$ , too.

( $\Leftarrow$ ) Let us consider  $\rho' \in P^M$ . There is  $\rho \in P$  such that for every negative literal  $\text{not } a \in body(\rho)$ ,  $a \notin M$ , and dropping all negative literals from  $\rho$  results in  $\rho'$ . If  $body(\rho') \subseteq M'$ , then  $body(\rho) \subseteq lit(\mathcal{U})$ . Thus,  $nup(aic(\rho)) \subseteq \mathcal{U}$ . It follows that  $head(aic(\rho)) \cap \mathcal{U} \neq \emptyset$ . Thus,  $head(\rho) \cap lit(\mathcal{U}) \neq \emptyset$ . Since  $head(\rho)$  consists of atoms and  $head(\rho') = head(\rho)$ ,  $head(\rho') \cap M' \neq \emptyset$ . That is,  $M' \models \rho'$  and, consequently,  $M' \models P^M$ .  $\square$

**Lemma 4.29.** *Let  $P$  be a simple disjunctive logic program. A set  $M$  of atoms is an answer set of  $P$  if and only if  $ua(M)$  is a justified weak repair for  $\langle \emptyset, aic(P) \rangle$ .*

**Proof:** ( $\Rightarrow$ ) Let  $M$  be an answer set of  $P$ . That is,  $M$  is a minimal set closed under the rules in the reduct  $P^M$ . By Lemma 4.28,  $\{+a \mid a \in M\} \cup \{-a \mid a \notin M\}$  is closed under  $aic(P)$ . Let  $\mathcal{U}'$  be a set of update actions such that  $\{-a \mid a \notin M\} \subseteq \mathcal{U}' \subseteq \{+a \mid a \in M\} \cup \{-a \mid a \notin M\}$ . We define  $M' = \{a \mid +a \in \mathcal{U}'\}$ . Then  $M' \subseteq M$ . By Lemma 4.28,  $M' \models P^M$ . Since  $M$  is an answer set of  $P$ ,  $M' = M$  and  $\mathcal{U}' = \mathcal{U}$ . It follows that  $\{+a \mid a \in M\} \cup \{-a \mid a \notin M\}$  is a minimal set closed under  $aic(P)$  and containing  $\{-a \mid a \notin M\}$ . Since  $ua(M) = \{+a \mid a \in M\}$  and  $ne(\emptyset, \emptyset \circ ua(M)) = \{-a \mid a \notin M\}$ , Theorem 4.7 implies that  $ua(M)$  is justified weak repair for  $\langle \emptyset, aic(P) \rangle$ .

( $\Leftarrow$ ) By Theorem 4.7,  $\{+a \mid a \in M\} \cup \{-a \mid a \notin M\}$  is a minimal set containing  $\{-a \mid a \notin M\}$  and closed under  $aic(P)$ . By Lemma 4.28,  $M$  is a model of  $P^M$ . Let  $M' \subseteq M$  be a model of  $P^M$ . Again by Lemma 4.28,  $\{+a \mid a \in M'\} \cup \{-a \mid a \notin M'\}$  is closed under  $aic(P)$ . It follows that  $\{+a \mid a \in M'\} \cup \{-a \mid a \notin M'\} = \{+a \mid a \in M\} \cup \{-a \mid a \notin M\}$ . Thus,  $M' = M$  and so,  $M$  is a minimal model of  $P^M$ , that is, an answer set of  $P$ .  $\square$

We now move on to results concerning upper bounds (membership) and derive the main results of this section.

**Lemma 4.30.** *Let  $\eta$  be a finite set of normal active integrity constraints and let  $\mathcal{U}$  be a finite set of update actions. There is a least set of update actions  $\mathcal{W}$  such that  $\mathcal{U} \subseteq \mathcal{W}$  and  $\mathcal{W}$  is closed under  $\eta$ . Moreover, this least set  $\mathcal{W}$  can be computed in polynomial time in the size of  $\eta$  and  $\mathcal{U}$ .*

**Proof:** We prove the result by demonstrating a bottom-up process computing  $\mathcal{W}$ . The process is similar to that applied when computing a least model of a Horn program. We start with  $\mathcal{W}_0 = \mathcal{U}$ . Assuming that  $\mathcal{W}_i$  has been computed, we identify in  $\eta$  every active integrity constraint  $r$  such that  $nup(r) \subseteq lit(\mathcal{W}_i)$ , and add the head of each such rule  $r$  to  $\mathcal{W}_i$ . We call the result  $\mathcal{W}_{i+1}$ . If  $\mathcal{W}_{i+1} = \mathcal{W}_i$ , we stop. It is straightforward to prove that the last set constructed in the process is closed under  $\eta$ , contains  $\mathcal{U}$ , and is contained in every set that is closed under  $\eta$  and contains  $\mathcal{U}$ . Moreover, the construction can be implemented to run in polynomial time.  $\square$

**Lemma 4.31.** *Let  $\eta$  be a finite set of normal active integrity constraints and let  $\mathcal{U}'$  and  $\mathcal{U}''$  be sets of update actions. The problem whether there is a set  $\mathcal{U}$  of update actions such that  $\mathcal{U}$  is closed under  $\eta$  and  $\mathcal{U}' \subseteq \mathcal{U} \subseteq \mathcal{U}''$  is in NP.*

**Proof:** Once we nondeterministically guess  $\mathcal{U}$ , checking all the required conditions can be implemented in polynomial time.  $\square$

**Lemma 4.32.** *Let  $\eta$  be a finite set of normal active integrity constraints,  $\mathcal{I}$  a database, and  $\mathcal{E}$  be a set of update actions. The problem whether there is a set  $\mathcal{E}' \subseteq \mathcal{E}$  of update actions such that  $\mathcal{I} \circ \mathcal{E}' \models \eta$  is in NP.*

**Proof:** Once we nondeterministically guess  $\mathcal{E}$ , checking all the required conditions can be implemented in polynomial time.  $\square$

**Theorem 4.33.** *Let  $\mathcal{I}$  be a database and  $\eta$  a set of normal active integrity constraints. Then checking if there exists a justified repair (justified weak repair, respectively) for  $\langle \mathcal{I}, \eta \rangle$  is an NP-complete problem.*

**Proof:** By Theorem 4.15, it is enough to prove the result for justified weak repairs.

(MEMBERSHIP) The following algorithm decides the problem: (1) Nondeterministically guess a consistent set of update actions  $\mathcal{E}$ . (2) Compute  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . (3) If  $\mathcal{E} \cap ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) \neq \emptyset$  return NO. Otherwise, compute the least set  $\mathcal{W}$  of update actions that is closed under  $\eta$  and contains  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ . (4) If  $\mathcal{W} = \mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ , then return YES. Otherwise, return NO. From an earlier observation, it follows that the algorithm runs in polynomial time. From Theorem 4.7, it follows that the algorithm is correct.

(HARDNESS) The problem of the existence of an answer set of a simple normal logic program  $P$  is NP-complete. By Theorem 4.15 and Lemma 4.29,  $P$  has an answer set if and only if there exists a justified weak repair for  $\langle \emptyset, aic(P) \rangle$ . Since  $aic(P)$  can be constructed in polynomial time in the size of  $P$ , the result follows.  $\square$

**Theorem 4.34.** *Let  $\mathcal{I}$  be a database and  $\eta$  a set of active integrity constraints. The problem of the existence of a justified weak repair for  $\langle \mathcal{I}, \eta \rangle$  is a  $\Sigma_2^P$ -complete problem.*

**Proof:** (MEMBERSHIP) The problem can be decided by a nondeterministic polynomial-time Turing Machine with an NP-oracle. Indeed, in the first step, one needs to guess (nondeterministically) a consistent set  $\mathcal{E}$  of update actions. Setting  $\mathcal{U} = \mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ , one needs to verify that

1.  $\mathcal{E} \cap ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \emptyset$
2.  $\mathcal{U}$  is closed under  $\eta$
3. for each  $\mathcal{U}'$  such that  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) \subseteq \mathcal{U}' \subseteq \mathcal{U}$  and  $\mathcal{U}'$  closed under  $\eta$ ,  $\mathcal{U}' = \mathcal{U}$  (by Lemma 4.31, one call to an NP-oracle suffices)

(HARDNESS) The problem of the existence of an answer set of a simple disjunctive logic program  $P$  is  $\Sigma_2^P$ -complete. By Lemma 4.29,  $P$  has an answer set if and only if there exists a justified weak repair for  $\langle \emptyset, aic(P) \rangle$ . Thus, the result follows.  $\square$

**Theorem 4.35.** *Let  $\mathcal{I}$  be a database and  $\eta$  a set of active integrity constraints. The problem of the existence of a justified repair for  $\langle \mathcal{I}, \eta \rangle$  is a  $\Sigma_2^P$ -complete problem.*

**Proof:** (MEMBERSHIP) The problem can be decided by a nondeterministic polynomial-time Turing Machine with an NP-oracle. Indeed, in the first step, one needs to guess (nondeterministically) a consistent set  $\mathcal{E}$  of update actions. Setting  $\mathcal{U} = \mathcal{E} \cup ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E})$ , one needs to verify that

1.  $\mathcal{E} \cap ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) = \emptyset$
2.  $\mathcal{U}$  is closed under  $\eta$



3. for each  $\mathcal{U}'$  such that  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{E}) \subseteq \mathcal{U}' \subseteq \mathcal{U}$  and  $\mathcal{U}'$  closed under  $\eta$ ,  $\mathcal{U}' = \mathcal{U}$  (by Lemma 4.31, one call to an NP-oracle suffices)
4. for each  $\mathcal{E}'$  such that  $\mathcal{E}' \subset \mathcal{E}$ ,  $\mathcal{I} \circ \mathcal{E}' \not\models \eta$  (By Lemma 4.32, one call to an NP-oracle suffices).

(HARDNESS) Since for the class of instances  $\langle \emptyset, aic(P) \rangle$  justified weak repairs coincide with justified repairs (Theorem 4.14), the result follows.  $\square$

## 4.7 Some Implications of the Results Obtained so far

We recall that given a database  $\mathcal{I}$  and a set  $\eta$  of active integrity constraints, the goal is to replace  $\mathcal{I}$  with  $\mathcal{I}'$  so that  $\mathcal{I}'$  satisfies  $\eta$ . The set of update actions needed to transform  $\mathcal{I}$  into  $\mathcal{I}'$  must at least be a repair for  $\langle \mathcal{I}, \eta \rangle$ . However, it should also obey preferences captured by the heads of constraints in  $\eta$ .

Let us denote by  $\mathbf{R}(\mathcal{I}, \eta)$  ( $\mathbf{FR}(\mathcal{I}, \eta)$ ,  $\mathbf{JR}(\mathcal{I}, \eta)$ , respectively) the class of repairs (founded repairs, justified repairs, respectively) for  $\langle \mathcal{I}, \eta \rangle$ . The results of the chapter imply that

$$\mathbf{JR}(\mathcal{I}, \eta^n) \subseteq \mathbf{JR}(\mathcal{I}, \eta) \subseteq \mathbf{FR}(\mathcal{I}, \eta) \subseteq \mathbf{R}(\mathcal{I}, \eta).$$

Thus, given an instance  $\langle \mathcal{I}, \eta \rangle$  of the database repair problem, one might first attempt to select a repair for  $\langle \mathcal{I}, \eta \rangle$  from the most restricted set of repairs —  $\mathbf{JR}(\mathcal{I}, \eta^n)$ . Not only these repairs are very strongly tied to preferences expressed by  $\eta$ , the related computational problems are relatively easy. The problem to decide whether this set is empty is NP-complete. However, the class  $\mathbf{JR}(\mathcal{I}, \eta^n)$  is narrow and it may be that  $\mathbf{JR}(\mathcal{I}, \eta^n) = \emptyset$ .

If it is so, as the next resort one might try to repair  $\mathcal{I}$  by selecting a repair from  $\mathbf{JR}(\mathcal{I}, \eta)$ . This class of repairs for  $\langle \mathcal{I}, \eta \rangle$  reflects the preferences captured by  $\eta$ . Since it is broader than the previous one, the chance of success is higher. However, the computational complexity grows — the existence problem for  $\mathbf{JR}(\mathcal{I}, \eta)$  is  $\Sigma_2^P$ -complete. If also  $\mathbf{JR}(\mathcal{I}, \eta) = \emptyset$ , it still may be that founded repairs exist. Moreover, deciding whether a founded repair exists is not harder than the previous step. Finally, if there are no founded repairs, one still may consider just a repair. This is not quite satisfactory as it ignores the preferences encoded by  $\eta$  and concentrates only on the constraint enforcement. However, deciding whether a repair exists is “only” NP-complete. Moreover, this class subsumes all other ones and so, the chance of success at this step is the largest.

We note that if we fail to find a justified or founded repair in the process described above, we may decide that respecting preferences encoded in active integrity constraints is more important than the minimality of change postulate. In such case, we have also an option to consider justified weak repairs of  $\langle \mathcal{I}, \eta \rangle$ , where the existence problem is  $\Sigma_2^P$ -complete and, then founded weak repairs for  $\langle \mathcal{I}, \eta \rangle$ , where the existence problem is NP-complete.

## 4.8 Connections between Revision Programs and Active Integrity Constraints

In this section we relate active integrity constraints to revision programs [108], an earlier formalism for expressing integrity constraints and prescribing preferred ways to enforce them.

### 4.8.1 Revision Programming — an Overview

A *revision literal* is an expression of the form  $\mathbf{in}(a)$  or  $\mathbf{out}(a)$ , where  $a$  is an atom ( $a \in At$ ). Revision literals  $\mathbf{in}(a)$  and  $\mathbf{out}(a)$  are *duals* of each other. If  $\alpha$  is a revision literal, we denote its dual by  $\alpha^D$ . We extend this notation to sets of revision literals. We say that a set of revision literals is *consistent* if it does not contain a pair of dual literals (or, in our notation, if  $\mathcal{U} \cap \mathcal{U}^D = \emptyset$ ).

Revision literals represent elementary updates one can apply to a database. We define the result of applying a *consistent* set  $\mathcal{U}$  of revision literals to a database  $\mathcal{I}$  as follows:

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

A *revision rule* is an expression of the form

$$r = \alpha_1 \vee \dots \vee \alpha_k \leftarrow \beta_1, \dots, \beta_m, \quad (4.4)$$

where  $k + m \geq 1$ , and  $\alpha_i$  and  $\beta_j$  are revision literals. A *revision program* is a collection of revision rules.

The set  $\{\alpha_1, \dots, \alpha_k\}$  is the *head* of the rule (4.4); we denote it by  $head(r)$ . Similarly, the set  $\{\beta_1, \dots, \beta_m\}$  is the *body* of the rule (4.4); we denote it by  $body(r)$ . If  $\vee head(r) \vee \leq 1$ , we call  $r$  a *normal* revision rule. Moreover, if  $|head(r)| = 0$ , we call  $r$  a *revision constraint*. Finally, a revision program is *normal* if all its rules are normal.

We say that a database  $d$  *satisfies* a revision literal  $\mathbf{in}(a)$  ( $\mathbf{out}(b)$ ), respectively, if  $a \in d$  ( $b \notin d$ , respectively). A database  $d$  *satisfies* a revision rule (4.4) if there is  $j$ ,  $1 \leq j \leq m$ , such that  $d$  does not satisfy  $\beta_j$ , or if there is  $i$ ,  $1 \leq i \leq k$ , such that  $d$  satisfies  $\alpha_i$ . Finally, a database  $d$  satisfies a revision program  $P$ , if  $d$  satisfies every rule in  $P$ . We use the symbol  $\models$  to denote the satisfaction relation. We often write “is a model of” instead of “satisfies.”

For a propositional literal  $L$ , if  $L = a$ , we define  $rl(L) = \mathbf{in}(a)$ . If  $L = \text{not } a$ , we define  $rl(L) = \mathbf{out}(a)$ . Conversely, for a revision literal  $\alpha = \mathbf{in}(a)$ , we set  $lit(\alpha) = a$  and for  $\alpha = \mathbf{out}(a)$ ,  $lit(\alpha) = \text{not } a$ . Finally, we extend the notation introduced here to sets of literals and sets of revision literals, as appropriate.

We note that every database interpretes revision literals and the corresponding propositional literals in the same way.

**Proposition 4.36.** *Let  $\mathcal{I}$  be a database. Then, for every set  $L$  of revision literals,  $\mathcal{I} \models L$  if and only if  $\mathcal{I} \models lit(L)$ .  $\square$*

It follows that with respect to this satisfaction relation and the corresponding concept of a model, a revision rule (4.4) is simply an integrity constraint equivalent to the propositional formula:

$$\text{lit}(\beta_1), \dots, \text{lit}(\beta_m) \supset \text{lit}(\alpha_1), \dots, \text{lit}(\alpha_k).$$

However, a revision rule functions not only as an integrity constraint. It also encodes a preference on how to “fix” it, when it does not hold. Not satisfying a revision rule means satisfying all revision literals in the body of  $r$  and failing to satisfy any of the revision literals in the head of  $r$ . Thus, fixing the constraint means constructing a database that (1) does not satisfy some revision literal in the body of  $r$ , or (2) satisfies at least one revision literal in the head of  $r$ .

Let  $P_1$  be a revision program consisting of a rule  $\mathbf{out}(b) \leftarrow \mathbf{in}(a)$ , and let  $\mathcal{I} = \{a, b\}$  be a database. Clearly,  $\mathcal{I}$  does not satisfy  $P_1$ . The program  $P_1$  has three models:  $\{b\}$ ,  $\{a\}$  and  $\emptyset$ . The first model violates the body of the rule, the second one satisfies the head of the rule, the third one has both properties. These models can be obtained by updating  $\mathcal{I}$  with  $\mathcal{U}_1 = \{\mathbf{out}(a)\}$ ,  $\mathcal{U}_2 = \{\mathbf{out}(a)\}$  and  $\mathcal{U}_3 = \{\mathbf{out}(a), \mathbf{out}(b)\}$ , respectively.

**Definition 4.37.** (WEAK REVISIONS AND REVISIONS) *A set  $\mathcal{U}$  of revision literals is a weak revision of  $\mathcal{I}$  with respect to a revision program  $P$  if*

1.  $\mathcal{I} \cap \{a \mid \mathbf{in}(a) \in \mathcal{U}\} = \emptyset$  and  $\{a \mid \mathbf{out}(a) \in \mathcal{U}\} \subseteq \mathbf{i}$  (that is, all revision literals in  $\mathcal{U}$  actually change  $\mathcal{I}$ )
2.  $\mathcal{I} \oplus \mathcal{U} \models P$  (constraint enforcement)

*A set  $\mathcal{U}$  of revision literals is a revision of  $\mathcal{I}$  with respect to a revision program  $P$  if*

1.  $\mathcal{I} \oplus \mathcal{U} \models P$  (constraint enforcement)
2. for every  $\mathcal{U}' \subseteq \mathcal{U}$ ,  $\mathcal{I} \oplus \mathcal{U}' \models P$  implies that  $\mathcal{U}' = \mathcal{U}$  (minimality of change)

Due to the minimality of change requirement, revisions are weak revisions (that is, consist of “status-changing” literals only). Furthermore, we note that the sets  $\mathcal{U}_1$  and  $\mathcal{U}_2$  in the example considered above are revisions. The set  $\mathcal{U}_3$  is a weak revision but not a revision.

To narrow down the class of acceptable (weak) revisions, [108] proposed the semantics of *justified revisions*. Speaking informally, that semantics gives the *preference* to models of a revision rule that satisfy its head over those models that do not satisfy its body. Thus, in our example, the database  $\{a\}$  is preferred over the database  $\{b\}$ .

We will now present a formal definition of justified revisions. The original definition from [108] dealt with the case of normal revision programs and did not explicitly mention the minimality of change requirement (it was implicit in the definition). The extension to disjunctive revision programs [118] also did not require the minimality of change. In analogy with the semantics of active integrity constraints, in this chapter we call justified revisions of [108] and [118], justified *weak* revisions.

A set  $\mathcal{U}$  of revision literals is *closed* under  $P$  if for every rule  $r \in P$ , whenever  $\text{body}(r) \subseteq \mathcal{U}$ , then  $\text{head}(r) \cap \mathcal{U} \neq \emptyset$ . If  $\mathcal{U}$  is closed under  $P$  and for every set  $\mathcal{U}' \subseteq \mathcal{U}$  closed under  $P$ , we have  $\mathcal{U}' = \mathcal{U}$ , then  $\mathcal{U}$  is a *minimal closed set* for  $P$ .

If a revision program  $P$  has no revision constraints, minimal closed sets exist. In general, a revision program may have no closed sets and so, no minimal closed sets, either (cf. the program consisting of the following two rules:  $\leftarrow \mathbf{in}(a)$  and  $\mathbf{in}(a) \leftarrow$ ).

By itself, a minimal closed set for a revision program  $P$  is not sufficient to determine the change that needs to be applied to a database to ensure it satisfies  $P$ . For instance, the program  $P_1 = \{\mathbf{out}(b) \leftarrow \mathbf{in}(a)\}$  has exactly one minimal closed set, namely  $\emptyset$ . But applying it to  $\{a, b\}$  does not result in any change. Of course, it is to be expected. When determining changes to be made we must take into account the initial and the revised databases.

Let  $\mathcal{I}$  be a database and  $\mathcal{R}$  a result of revising  $\mathcal{I}$ . We define the *inertia set* with respect to  $\mathcal{I}$  and  $\mathcal{R}$ , denoted  $I(\mathcal{I}, \mathcal{R})$ , by setting

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

In other words,  $I(\mathcal{I}, \mathcal{R})$  is the set of all revision literals that have no effect when revising  $\mathcal{I}$  into  $\mathcal{R}$ . Thus, when using  $P$  to justify a transformation from  $\mathcal{I}$  to  $\mathcal{R}$ , we may assume all revision literals in  $I(\mathcal{I}, \mathcal{R})$ .

**Definition 4.38.** (JUSTIFIED UPDATES AND JUSTIFIED WEAK REVISIONS)

Let  $P$  be a revision program and let  $\mathcal{I}$  be a database. A set  $\mathcal{U}$  of revision literals is a  $P$ -justified update for  $\mathcal{I}$  if

1.  $\mathcal{U}$  is consistent, and
2.  $\mathcal{U}$  is a minimal set closed under  $P \cup I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$ .

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

While not self-evident from the definition, justified weak updates and justified weak revisions, when applied to an initial database yield a database satisfying the program (cf. [108, 118]).

For normal revision programs, justified weak revisions are minimal [108]. However, in general, the condition (2) in Definition 4.38 is insufficient to enforce the minimality of  $P$ -justified weak revisions. Let  $P_2 = \{\mathbf{out}(a) \vee \mathbf{in}(a) \leftarrow\}$  and let  $\mathcal{I} = \emptyset$ . One can check that both  $\mathcal{E}_1 = \emptyset$  and  $\mathcal{E}_2 = \{\mathbf{in}(a)\}$  are  $P$ -justified weak revisions for  $\mathcal{I}$ .

**Definition 4.39.** (JUSTIFIED REVISIONS)

Let  $P$  be a revision program and let  $\mathcal{I}$  be a database. A  $P$ -justified weak revision  $\mathcal{E}$  for  $\mathcal{I}$  is a  $P$ -justified revision for  $\mathcal{I}$  if for every set  $\mathcal{E}' \subseteq \mathcal{E}$  such that  $\mathcal{I} \oplus \mathcal{E}' \models P$ ,  $\mathcal{E}' = \mathcal{E}$ .  $\square$

As we will see, justified (weak) revisions correspond to justified (weak) repairs. We will now introduce a new semantics for revision programs motivated by intuitions behind the semantics of founded repairs of active integrity constraints.

**Definition 4.40.** (FOUNDED (WEAK) REVISIONS) *Let  $\mathcal{I}$  be a database,  $P$  a revision program and, and  $\mathcal{E}$  a consistent set of revision literals.*

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

There are examples showing that, in general, (weak) revisions are not founded (weak) revisions, and founded weak revisions are not founded revisions.

*Example 4.41.* Let  $\mathcal{I} = \emptyset$  and  $P$  be the revision program containing the following revision rules:

$$\begin{aligned} r_1 &= \mathbf{in}(c) \leftarrow \mathbf{out}(d) \\ r_2 &= \mathbf{in}(b) \leftarrow \mathbf{in}(a) \\ r_3 &= \mathbf{in}(a) \leftarrow \mathbf{in}(b) \end{aligned}$$

The set  $\{\mathbf{in}(d)\}$  is a revision of  $\mathcal{I}$  with respect to  $P$ . Therefore it is a weak revision of  $\mathcal{I}$  with respect to  $P$ . However it is not a  $P$ -founded weak revision for  $\mathcal{I}$ . Therefore, it is not a  $P$ -founded revision for  $\mathcal{I}$ . The set  $\{\mathbf{in}(c), \mathbf{in}(a), \mathbf{in}(b)\}$  is a  $P$ -founded weak revision for  $\mathcal{I}$  but not a  $P$ -founded revision for  $\mathcal{I}$ .  $\square$

**Proposition 4.42.** *Let  $P$  be a revision program and let  $\mathcal{I}$  be a database. If  $\mathcal{E}$  is a  $P$ -justified weak revision of  $\mathcal{I}$ , then it is a  $P$ -founded weak revision of  $\mathcal{I}$ .*

**Proof:** We know that  $\mathcal{I} \oplus \mathcal{E} \models P$  (cf. [108, 118]) that is  $\mathcal{E}$  is a weak revision of  $\mathcal{I}$  with respect to  $P$ . Therefore, we need to prove that  $\mathcal{E}$  is  $P$ -founded with respect to  $\mathcal{I}$ . As  $\mathcal{E}$  is a  $P$ -justified weak revision of  $\mathcal{I}$  there exists a  $P$ -justified update  $\mathcal{U}$  of  $\mathcal{I}$  such that  $\mathcal{E} = \mathcal{U} \setminus I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$ . We know that  $\mathcal{U}$  is consistent and is a minimal set closed under  $P \cup I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$ . Let  $\alpha \in \mathcal{E} \subseteq \mathcal{U}$ . As  $\mathcal{U}$  is minimal,  $\mathcal{U}' = \mathcal{U} \setminus \{\alpha\}$  is not closed under  $P \cup I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$ . As  $\alpha \notin I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$  there must be a revision rule  $r \in P$  such that  $\text{body}(r) \subseteq \mathcal{U}'$  and  $\text{head}(r) \cap \mathcal{U}' = \emptyset$ . As  $\mathcal{U}' \subseteq \mathcal{U}$  we have that  $\text{body}(r) \subseteq \mathcal{U}$ . Therefore, as  $\mathcal{U}$  is closed under  $r$ ,  $\text{head}(r) \cap \mathcal{U} = \{\alpha\}$ . It follows that  $\mathcal{I} \oplus \mathcal{U} = \mathcal{I} \oplus \mathcal{E} \models \text{body}(r)$  and for each  $\beta \in \text{head}(r) \setminus \{\alpha\}$  we have that  $\beta \notin \mathcal{U}$ . Therefore,  $\beta \notin \mathcal{E}$  and  $\beta \notin I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$ . We have two cases: either  $\mathcal{I} \models \beta$  or  $\mathcal{I} \not\models \beta$ . In the first case as  $\beta \notin I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$  then  $\beta^D \in \mathcal{E}$  while in the second case as  $\beta \notin \mathcal{E}$  we have that  $\mathcal{I} \oplus \mathcal{E} \not\models \beta$ . In each case  $\mathcal{I} \oplus \mathcal{E} \models \beta^D$ .  $\square$

**Proposition 4.43.** *Let  $P$  be a revision program and let  $\mathcal{I}$  be a database. If  $\mathcal{E}$  is a  $P$ -justified revision of  $\mathcal{I}$ , then it is a  $P$ -founded revision of  $\mathcal{I}$ .*

**Proof:** We know that  $\mathcal{E}$  is a  $P$ -justified weak revision of  $\mathcal{I}$ . Therefore, it is a weak revision of  $\mathcal{I}$  with respect to  $P$ . Moreover as  $\mathcal{E}$  is a  $P$ -justified revision of  $\mathcal{I}$  for every

set  $\mathcal{E}' \subseteq \mathcal{E}$  such that  $\mathcal{I} \oplus \mathcal{E}' \models P$ ,  $\mathcal{E}' = \mathcal{E}$ . Thus  $\mathcal{E}$  is a revision of  $\mathcal{I}$  with respect to  $P$ . As by Proposition 4.42 we have that  $\mathcal{E}$  is a  $P$ -founded weak revision of  $\mathcal{I}$  it follows that  $\mathcal{E}$  is a  $P$ -founded revision of  $\mathcal{I}$ .  $\square$

The converse implications do not hold in general. Let  $P = \{\mathbf{in}(b) \leftarrow \mathbf{in}(a), \mathbf{in}(b) \leftarrow \mathbf{out}(a), \mathbf{in}(a) \leftarrow \mathbf{in}(b)\}$  and let  $\mathcal{I} = \emptyset$ . One can check that  $\mathcal{R} = \{\mathbf{in}(a), \mathbf{in}(b)\}$  is a  $P$ -founded revision of  $\mathcal{I}$  (and so, a  $P$ -founded weak revision of  $\mathcal{I}$ , too). However, it is not a  $P$ -justified weak revision of  $\mathcal{I}$  (and so, also not a  $P$ -justified revision of  $\mathcal{I}$ ).

To summarize our discussion, revision programs can be assigned the semantics of (weak) revisions, justified (weak) revisions and founded (weak) revisions. The similarities to active integrity constraints are striking. We will establish the precise connection in the next two sections.

### 4.8.2 Proper Revision Programs

To relate revision programs and active integrity constraints, we first note that we can restrict the syntax of revision programs without affecting their expressivity.

A *proper revision rule* is a revision rule that satisfies the following condition: the literal in the head of the rule is not the dual of any literal in the body of the rule.

Let  $P$  be a revision program with constraints and let  $r_1$  and  $r_2$  be revision rules

$$\alpha \vee \alpha_1 \vee \dots \vee \alpha_k \leftarrow \alpha^D, \beta_1, \dots, \beta_m$$

and

$$\alpha_1 \vee \dots \vee \alpha_k \leftarrow \alpha^D, \beta_1, \dots, \beta_m,$$

respectively (that is,  $r_2$  differs from  $r_1$  in that it drops  $\alpha$  from the head).

**Lemma 4.44.** *Let  $\mathcal{I}$  be a database. Under the notation introduced above, a set of revision literals is a (weak) revision of  $\mathcal{I}$  with respect to  $P \cup \{r_1\}$  ( $P \cup \{r_1\}$ -founded (weak) revision,  $P \cup \{r_1\}$ -justified (weak) revision of  $\mathcal{I}$ , respectively) if and only if  $\mathcal{U}$  is a (weak) revision of  $\mathcal{I}$  with respect to  $P \cup \{r_2\}$  ( $P \cup \{r_2\}$ -founded (weak) revision,  $P \cup \{r_2\}$ -justified (weak) revision of  $\mathcal{I}$ , respectively).*

**Proof:** The claim is evident for the case of weak revisions and revisions. The case of justified (weak) revisions follows from the observation that a consistent set  $\mathcal{U}$  of revision literals is a closed set for  $P \cup \{r_1\} \cup I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$  if and only if  $\mathcal{U}$  is a closed set for  $P \cup \{r_2\} \cup I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$ .

For the case of founded (weak) revisions, it is enough to prove that a set  $\mathcal{U}$  of revision literals is founded with respect to  $\langle \mathcal{I}, P \cup \{r_1\} \rangle$  if and only if  $\mathcal{U}$  is founded with respect to  $\langle \mathcal{I}, P \cup \{r_2\} \rangle$ . Let  $\beta \in \mathcal{U}$  be founded with respect to  $\langle \mathcal{I}, P \cup \{r_1\} \rangle$  and  $\mathcal{U}$ , and let  $r \in P \cup \{r_1\}$  be the rule providing support to  $\beta$ . If  $r \neq r_1$ ,  $r \in P$  and so,  $\beta$  is founded with respect to  $\langle \mathcal{I}, P \cup \{r_2\} \rangle$  and  $\mathcal{U}$ . Thus, let us assume that  $r = r_1$ . If  $\beta = \alpha$ , then  $\alpha \in \mathcal{U}$ ,  $\mathcal{I} \circ \mathcal{U} \models \alpha$ . Since  $\mathcal{I} \circ \mathcal{U} \models \text{body}(r_1)$ ,  $\mathcal{I} \circ \mathcal{U} \models \alpha^D$ , a contradiction. Thus,  $\beta \neq \alpha$ . It is easy to see that in such case,  $r_2$  supports  $\beta$  (given  $\mathcal{U}$ ). Thus,  $\beta$  is founded with respect to  $\langle \mathcal{I}, P \cup \{r_2\} \rangle$  in this case, too.

Conversely, let  $\beta \in \mathcal{U}$  be founded with respect to  $\langle \mathcal{I}, P \cup \{r_2\} \rangle$  and  $\mathcal{U}$ , and let  $r \in P \cup \{r_2\}$  be the rule providing support to  $\alpha$ . As before, if  $r \neq r_2$ , the claim follows. If  $r = r_2$ , then  $\beta \neq \alpha$ . Since  $r_2$  supports  $\beta$ , one can check that  $r_1$  supports,  $\beta$ , too.  $\square$

**Theorem 4.45.** *Let  $P$  be a revision program. There is a proper revision program  $P'$  such that for every database  $\mathcal{I}$ , (weak) revisions of  $\mathcal{I}$  with respect to  $P$  ( $P$ -founded (weak) revisions,  $P$ -justified (weak) revisions of  $\mathcal{I}$ , respectively) coincide with (weak) revisions of  $\mathcal{I}$  with respect to  $P'$  ( $P'$ -founded (weak) revisions,  $P'$ -justified (weak) revisions of  $\mathcal{I}$ , respectively).*

**Proof:** Lemma 4.44 implies that the program  $P'$  obtained from  $P$  by repeated application of the process described above (replacement of rules of the form  $r_1$  with the corresponding rules of the form  $r_2$ ) has the required property.  $\square$

### 4.8.3 Revision Programs as Sets of Active Integrity Constraints

**Definition 4.46.** *Given a revision rule  $r$  of the form*

$$\alpha_1 \vee \dots \vee \alpha_k \leftarrow \beta_1, \dots, \beta_m$$

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

$$lit(\beta_1), \dots, lit(\beta_m), lit(\alpha_1)^D, \dots, lit(\alpha_k)^D \supset ua(\alpha_1) \vee \dots \vee ua(\alpha_k). \quad \square$$

We note that if  $r$  is a revision constraint ( $k = 0$ ),  $AIC(r)$  is simply an integrity constraint. The operator  $AIC(\cdot)$  is extended to revision programs in the standard way. It is easy to show that for each database  $d$ ,  $d \models P$  if and only if  $d \models AIC(P)$ . The following lemma establishes a direct connection between the concepts of closure under active integrity constraints and revision programs.

**Theorem 4.47.** *Let  $P$  be a proper revision program. A set  $\mathcal{E}$  of revision literals is a weak revision of  $\mathcal{I}$  with respect to  $P$  if and only if  $ua(\mathcal{E})$  is a weak repair for  $\langle \mathcal{I}, AIC(P) \rangle$ .*

**Proof:** By the definition,  $\mathcal{E}$  is a weak revision of  $\mathcal{I}$  with respect to  $P$  if and only if

1.  $\mathcal{I} \cap \{a \mid \mathbf{in}(a) \in \mathcal{E}\} = \emptyset$ ,  $\{a \mid \mathbf{out}(a) \in \mathcal{E}\} \subseteq \mathcal{I}$ ; and
2.  $\mathcal{I} \oplus \mathcal{E} \models P$ .

Similarly,  $ua(\mathcal{E})$  is a weak repair for  $\langle \mathcal{I}, AIC(P) \rangle$  if and only if

1.  $\mathcal{I} \cap \{a \mid +a \in ua(\mathcal{E})\} = \emptyset$ ,  $\{a \mid -a \in ua(\mathcal{E})\} \subseteq \mathcal{I}$ ; and
2.  $\mathcal{I} \circ ua(\mathcal{E}) \models AIC(P)$ .

By our earlier comments, for every database  $\mathcal{DB}$ ,  $\mathcal{DB} \models P$  if and only if  $\mathcal{DB} \models AIC(P)$ . Since  $\mathcal{I} \oplus \mathcal{E} = \mathcal{I} \circ ua(\mathcal{E})$ , the assertion follows.  $\square$

**Theorem 4.48.** *Let  $P$  be a proper revision program. A set of revision literals  $\mathcal{E}$  is a revision of  $\mathcal{I}$  if and only if  $ua(\mathcal{E})$  is a repair for  $\langle \mathcal{I}, AIC(P) \rangle$ .*

**Proof:** By Theorem 4.47, we have that  $\mathcal{E}$  is a weak revision of  $\mathcal{I}$  if and only if  $ua(\mathcal{E})$  is a weak repair for  $\langle \mathcal{I}, AIC(P) \rangle$ . Moreover,  $\mathcal{E}$  is such that for each  $\mathcal{E}' \subseteq \mathcal{E}$  the fact  $\mathcal{I} \oplus \mathcal{E}' \models P$  implies  $\mathcal{E}' = \mathcal{E}$  if and only if  $ua(\mathcal{E})$  is such that for each  $ua(\mathcal{E}') \subseteq ua(\mathcal{E})$  the fact  $\mathcal{I} \circ ua(\mathcal{E}') \models P$  (that is  $\mathcal{I} \circ ua(\mathcal{E}') \models AIC(P)$ ) implies  $ua(\mathcal{E}') = ua(\mathcal{E})$ .  $\square$

**Lemma 4.49.** *Let  $r$  be a proper revision rule. A set  $\mathcal{E}$  of revision literals is closed under  $P$  if and only if  $ua(\mathcal{E})$  is closed under  $AIC(r)$ .*

**Proof:** First, we observe that as  $r$  is proper,  $nup(AIC(r)) = lit(body(r))$ . Moreover  $head(AIC(r)) = ua(head(r))$ . We know that  $\mathcal{E}$  is closed under  $r$  if and only if  $body(r) \not\subseteq \mathcal{E}$  or  $head(r) \cap \mathcal{E} \neq \emptyset$ . This holds if and only if  $lit(body(r)) \not\subseteq lit(\mathcal{E}) = lit(ua(\mathcal{E}))$  or  $ua(head(r)) \cap ua(\mathcal{E}) \neq \emptyset$ , which is equivalent to  $nup(AIC(r)) \not\subseteq lit(ua(\mathcal{E}))$  or  $head(AIC(r)) \cap ua(\mathcal{E}) \neq \emptyset$ . This, however, is the definition of  $AIC(r)$  closed under  $ua(\mathcal{E})$ .  $\square$

**Corollary 4.50.** *Let  $P$  be a proper revision program. A set  $\mathcal{E}$  of revision literals is a minimal set closed under  $P$  if and only if  $ua(\mathcal{E})$  is a minimal set closed under  $AIC(r)$ .*

**Proof:** Straightforward from Lemma 4.49.  $\square$

**Theorem 4.51.** *Let  $P$  be a proper revision program. A set of revision literals  $\mathcal{E}$  is a  $P$ -justified weak revision of  $\mathcal{I}$  if and only if  $ua(\mathcal{E})$  is a justified weak repair for  $\langle \mathcal{I}, AIC(P) \rangle$ .*

**Proof:** ( $\Rightarrow$ ) The set  $\mathcal{E}$  is a  $P$ -justified revision of  $\mathcal{I}$  and so it is a  $P$ -justified weak revision of  $\mathcal{I}$ . Therefore, there exists a  $P$ -justified weak update of  $\mathcal{I}$ , say  $\mathcal{U}$ , s.t.  $\mathcal{E} = \mathcal{U} \setminus I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$ . By definition,  $\mathcal{U}$  is consistent and it is a minimal set containing  $I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})$  and closed under  $P$ . It follows that the action set  $ua(\mathcal{U})$  is consistent and, by Corollary 4.50, it is a minimal set containing  $ua(I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U}))$  and closed under  $AIC(P)$ . Now we observe that  $ua(I(\mathcal{I}, \mathcal{I} \oplus \mathcal{U})) = ne(\mathcal{I}, \mathcal{I} \circ ua(\mathcal{U}))$ . Thus,  $ua(\mathcal{U})$  is a justified action set for  $\langle \mathcal{I}, AIC(P) \rangle$  and  $ua(\mathcal{U}) \setminus ne(\mathcal{I}, \mathcal{I} \circ ua(\mathcal{U})) = ua(\mathcal{E})$  is a justified weak repair for  $\langle \mathcal{I}, AIC(P) \rangle$ .

( $\Leftarrow$ ) The set  $ua(\mathcal{E})$  is a justified repair for  $\langle \mathcal{I}, AIC(P) \rangle$  and so it is a justified weak repair for  $\langle \mathcal{I}, AIC(P) \rangle$ . Thus, there exists a justified action set for  $\langle \mathcal{I}, AIC(P) \rangle$ , say  $\mathcal{U}$ , s.t.  $ua(\mathcal{E}) = \mathcal{U} \setminus ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ . The action set  $\mathcal{U}$  is consistent, contains  $ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$  and it is closed under  $AIC(P)$ . It follows that the set  $rl(\mathcal{U})$  is consistent and, by Corollary 4.50, it is a minimal set containing  $rl(ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U}))$  and closed under  $P$ . Now we observe that  $rl(ne(\mathcal{I}, \mathcal{I} \circ \mathcal{U})) = I(\mathcal{I}, \mathcal{I} \oplus rl(\mathcal{U}))$ . Thus,  $rl(\mathcal{U})$  is a  $P$ -justified weak update for  $\mathcal{I}$  and  $rl(\mathcal{U}) \setminus I(\mathcal{I}, \mathcal{I} \oplus rl(\mathcal{U})) = \mathcal{E}$  is a  $P$ -justified weak revision for  $\mathcal{I}$ .  $\square$

**Theorem 4.52.** *Let  $P$  be a proper revision program. A set of revision literals  $\mathcal{E}$  is a  $P$ -justified revision of  $\mathcal{I}$  if and only if  $ua(\mathcal{E})$  is a justified repair for  $\langle \mathcal{I}, AIC(P) \rangle$ .*



**Proof:** ( $\Rightarrow$ ) The set  $\mathcal{E}$  is a  $P$ -justified revision of  $\mathcal{I}$  and so it is a  $P$ -justified weak revision of  $\mathcal{I}$ . By Theorem 4.51,  $ua(\mathcal{E})$  is a justified weak repair for  $\langle \mathcal{I}, AIC(P) \rangle$ . As  $\mathcal{E}$  is a  $P$ -justified revision of  $\mathcal{I}$ , for every set  $\mathcal{E}' \subseteq \mathcal{E}$  such that  $\mathcal{I} \oplus \mathcal{E}' \models P$ ,  $\mathcal{E}' = \mathcal{E}$ . As  $\mathcal{I} \oplus \mathcal{E}' = \mathcal{I} \circ ua(\mathcal{E}')$  and for each database  $\mathcal{DB}$ ,  $\mathcal{DB} \models P$  if and only if  $\mathcal{DB} \models AIC(P)$ , we have that for each  $ua(\mathcal{E}') \subseteq ua(\mathcal{E})$  such that  $\mathcal{I} \circ ua(\mathcal{E}') \models AIC(P)$ ,  $ua(\mathcal{E}') = ua(\mathcal{E})$  that is  $ua(\mathcal{E})$  is a justified repair for  $\langle \mathcal{I}, AIC(P) \rangle$ .

( $\Leftarrow$ ) The set  $ua(\mathcal{E})$  is a justified repair for  $\langle \mathcal{I}, AIC(P) \rangle$  and so it is a justified weak repair for  $\langle \mathcal{I}, AIC(P) \rangle$ . By Theorem 4.51,  $\mathcal{E}$  is a  $P$ -justified weak revision for  $\mathcal{I}$ . Following a reasoning similar to that of part ( $\Rightarrow$ ), it can be proved the minimality of  $\mathcal{E}$  i.e. that  $\mathcal{E}$  is a  $P$ -justified revision for  $\mathcal{I}$ .  $\square$

The following theorem establishes the correspondence between founded (weak) revisions and founded (weak) repairs.

**Theorem 4.53.** *Let  $P$  be a proper revision program. A set of revision literals  $\mathcal{E}$  is a  $P$ -founded (weak) revision of  $\mathcal{I}$  if and only if  $ua(\mathcal{E})$  is a founded (weak) repair for  $\langle \mathcal{I}, AIC(P) \rangle$ .*

**Proof:** ( $\Rightarrow$ ) Let  $\mathcal{E}$  be a  $P$ -founded (weak) revision of  $\mathcal{I}$ . By Theorem 4.47,  $ua(\mathcal{E})$  is a (weak) repair for  $\langle \mathcal{I}, AIC(P) \rangle$ . Therefore we have to show that  $ua(\mathcal{E})$  is founded with respect to  $\langle \mathcal{I}, AIC(P) \rangle$ . Let us consider an arbitrary element of  $ua(\mathcal{E})$ . It is of the form  $ua(\alpha)$ , for some revision literal  $\alpha \in \mathcal{E}$ .

Since  $\mathcal{E}$  is  $P$ -founded with respect to  $\mathcal{I}$ , there exists  $r \in P$  such that  $\mathcal{I} \oplus \mathcal{E} \models body(r)$ , and  $\mathcal{I} \oplus \mathcal{E} \models \gamma^D$ , for every  $\gamma \in head(r)$  different from  $\alpha$ . Let  $\rho$  be the corresponding active integrity constraint in  $AIC(P)$ , that is,  $\rho = AIC(r)$ . Since  $r$  is proper,  $lit(body(r)) = nup(\rho)$ . Thus,  $\mathcal{I} \circ ua(\mathcal{E}) \models nup(\rho)$ . Moreover, since  $head(\rho) = ua(head(r))$ , for every  $\delta \in head(\rho)$  other than  $ua(\alpha)$ ,  $\mathcal{I} \circ ua(\mathcal{E}) \models \delta^D$ . Thus,  $ua(\alpha)$  is founded with respect to  $\langle \mathcal{I}, AIC(P) \rangle$  and  $ua(\mathcal{E})$  and so,  $ua(\mathcal{E})$  is founded with respect to  $\langle \mathcal{I}, AIC(P) \rangle$ .

( $\Leftarrow$ ) This implication can be proved by a similar argument. We omit the details.  $\square$

The results of this section show that proper revision programs can be interpreted as sets of active integrity constraints so that the corresponding semantics match. However, it is easy to see that the mapping  $AIC(\cdot)$  is a one-to-one and onto mapping between the collection of revision programs and the collections of sets of active integrity constraints. Thus, also conversely, sets of active integrity constraints can be interpreted as revision programs.

#### 4.8.4 Shifting Theorem for Revision Programs

The concept of “shifting” presented in Section 4.5 can be reformulated for revision programming. Many results about shifting properties of revision programs are presented in [109]. In this section we derive these and further results indirectly from the shifting properties of active integrity constraints using the equivalence results presented in Section 5.12. The operator  $T_{\mathcal{W}}(\cdot)$  presented in Section 4.5 can be extended to revision literals, revision rules and revision programs. Its formal definition

and many properties are presented in [109]. Here we present further properties and use them to establish the shifting theorem for revision programs.

**Proposition 4.54.** *Let  $\mathcal{I}$  and  $\mathcal{W}$  be databases,  $\mathcal{E}$  a set of revision literals,  $G$  a revision program and  $P$  a proper revision program. Then  $T_{\mathcal{W}}(\text{prop}(G)) = \text{prop}(T_{\mathcal{W}}(G))$ ,  $T_{\mathcal{W}}(\text{ua}(\mathcal{E})) = \text{ua}(T_{\mathcal{W}}(\mathcal{E}))$  and  $T_{\mathcal{W}}(\text{AIC}(P)) = \text{aic}(T_{\mathcal{W}}(P))$ .*

**Proof.** Straightforward from the definitions of  $\text{prop}(\cdot)$ ,  $T_{\mathcal{W}}(\cdot)$ ,  $\text{ua}(\cdot)$  and  $\text{AIC}(\cdot)$ .  $\square$

**Theorem 4.55.** (SHIFTING THEOREM FOR REVISION PROGRAMS) *Let  $\mathcal{I}$  and  $\mathcal{W}$  be databases. For every revision program  $G$  and every consistent set  $\mathcal{E}$  of revision literals, we have*

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

**Proof:** Let  $P = \text{prop}(G)$  (that is the “properized” version of  $G$ ). The following properties are equivalent:

1.  $\mathcal{E}$  is a (weak) revision for  $\mathcal{I}$  with respect to  $G$  (respectively,  $G$ -justified (weak) revision for  $\mathcal{I}$ ,  $G$ -founded (weak) revision for  $\mathcal{I}$ )
2.  $\mathcal{E}$  is a (weak) revision for  $\mathcal{I}$  with respect to  $P$  (respectively,  $P$ -justified (weak) revision for  $\mathcal{I}$ ,  $P$ -founded (weak) revision for  $\mathcal{I}$ )
3.  $\text{ua}(\mathcal{E})$  is a (weak) repair (respectively, justified (weak) repair, founded (weak) repair) for  $\langle \mathcal{I}, \text{AIC}(P) \rangle$
4.  $T_{\mathcal{W}}(\text{ua}(\mathcal{E}))$  is a (weak) repair (respectively, justified (weak) repair, founded (weak) repair) for  $\langle \mathcal{I} \div \mathcal{W}, T_{\mathcal{W}}(\text{AIC}(P)) \rangle$
5.  $T_{\mathcal{W}}(\mathcal{E})$  is a (weak) revision for  $\mathcal{I} \div \mathcal{W}$  with respect to  $T_{\mathcal{W}}(P)$  (respectively,  $T_{\mathcal{W}}(P)$ -justified (weak) revision for  $\mathcal{I} \div \mathcal{W}$ ,  $T_{\mathcal{W}}(P)$ -founded (weak) revision for  $\mathcal{I} \div \mathcal{W}$ )
6.  $T_{\mathcal{W}}(\mathcal{E})$  is a (weak) revision for  $\mathcal{I} \div \mathcal{W}$  with respect to  $T_{\mathcal{W}}(G)$  (respectively,  $T_{\mathcal{W}}(G)$ -justified (weak) revision for  $\mathcal{I} \div \mathcal{W}$ ,  $T_{\mathcal{W}}(G)$ -founded (weak) revision for  $\mathcal{I} \div \mathcal{W}$ ).

Indeed, (1) and (2) are equivalent by Theorem 4.45, (2) and (3) are equivalent by Theorems 4.47 - 4.53, (3) and (4) — by Theorems 4.24 and 4.26. Next, (4) and (5) are equivalent by Theorems 4.47 - 4.53, as well as Proposition 4.54, and (5) and (6) — by Theorem 4.45 and Proposition 4.54. Thus, the assertion follows.  $\square$

## 4.9 Computation and Complexity Results for Revision Programming

Thanks to the equivalence properties reported in Section 5.12 we can derive the results about computation and complexity for revision programming from the corresponding results for active integrity constraints presented in Section 4.6.

Let  $\rho$  be a logic programming rule, say

$$\rho = a_1 \vee \dots \vee a_k \leftarrow \beta.$$

We define

$$rp(\rho) = rl(a_1) \vee \dots \vee rl(a_k) \leftarrow rl(\beta).$$

We extend the operator  $rp(\cdot)$  to logic programs in a standard way. We observe that if a logic program  $P$  is simple then the corresponding revision program  $rp(P)$  is proper.

**Lemma 4.56.** *Let  $P$  be a simple disjunctive logic program. A set  $M$  of atoms is an answer set of  $P$  if and only if  $rl(M)$  is a  $rp(P)$ -justified weak revision for  $\emptyset$ .*

**Proof.** Straightforward from Lemma 4.29 and Theorem 4.51. □

**Theorem 4.57.** *Let  $\mathcal{I}$  be a database and  $P$  a normal proper revision program. Then checking if there exists a  $P$ -justified revision ( $P$ -justified weak revision, respectively) for  $\mathcal{I}$  is an NP-complete problem.*

**Proof.** Straightforward from Theorems 4.33, 4.51 and 4.52. □

**Theorem 4.58.** *Let  $\mathcal{I}$  be a database and  $P$  a proper revision program. Then checking if there exists a  $P$ -justified revision ( $P$ -justified weak revision, respectively) for  $\mathcal{I}$  is a  $\Sigma_2^P$ -complete problem.*

**Proof.** Straightforward from Theorems 4.34, 4.35, 4.51 and 4.52. □

**Theorem 4.59.** *Let  $\mathcal{I}$  be a database and  $P$  a proper revision program. Then checking if there exists a  $P$ -founded revision ( $P$ -founded weak revision, respectively) for  $\mathcal{I}$  is a  $\Sigma_2^P$ -complete (NP-complete, respectively) problem.*

**Proof.** Straightforward from complexity results in [29] and Theorem 4.53. □



## View Updating through Active Integrity Constraints

**Summary.** This chapter presents a declarative semantics for view updating in the presence of existentially derived predicates and non-flat integrity constraints, that translates an update request against a view into an update of the underlying database. The novelty of the framework consists in the definition of a formal declarative semantics for view updating that allows to identify, among the set of all possible repairs, the subset of *supported repairs*, that is repairs whose actions are validated by the database or by other updates. Given a deductive database and an update request, the computation of supported repairs is performed by rewriting the update request and the deductive database in the form of active integrity constraints. The proposed approach will be shown to prevent the anomalies previous approaches suffer from, limiting the wide range of translations to those that are justified by the deductive database.

### 5.1 Introduction

Current database systems are often large and complex and the case that a user or an application has full access to the entire database is rare. It is more likely to occur that access is granted via windows of the entire systems, called *views*. A view, usually virtual, is defined by giving a query on the whole database and at any point the content of the view is just the outcome of this query. Applications query a base relation or a view in the same way. Therefore, querying a view does not represent a serious conceptual problem. In contrast, the issue of *view updating* is problematic and of paramount importance: it refers to the problem of translating an update request against a view into an update request involving the base of data. The basic problem underlying view updating is that a translation from a view update into corresponding updates over the extensional database does not always exist or several translations could be performed in order to satisfy the update request. The complexity of the view update problem arises even in the case of a simple update operation, such as inserting a tuple in a view. Current commercial DBMS, e.g. Access, MySQL, Oracle, SQLServer accept an update against a view, and propagate it to the stored relation, only in the simple case in which the view is defined from one database relation, and reject any update request against a view if this is defined by joining more than one re-

lation. This rigid behavior ensures the acceptance of an update request if and only if a unique translation exists and solves, albeit drastically, the ambiguity of more translations. This chapter focuses on view updating in the presence of *existentially derived predicates*<sup>1</sup> and *non-flat integrity constraints*<sup>2</sup> and proposes a logic framework that translates a view updating into an update of the underlying database. Specifically, given a deductive database consisting of a set of base facts, a set of integrity constraints and a set of deductive rules and given an update request, consisting of a set of insert and delete operations of base and derived facts, it allows us to determine how the update request can be translated into a minimal set of updates of the stored base facts, while ensuring integrity constraint maintenance and performing “smaller change”. The benefits of this proposal, evident in the presence of existential derived predicates, will be intuitively introduced by a few examples.

*Example 5.1.* Consider the update request  $+P(a)$  asking for the *insertion of the fact*  $P(a)$  and the deductive database

$$Q(a, b). \quad r_1 : P(X) \leftarrow Q(X, Y), R(X, Y, Z)$$

The request, not allowed by commercial DBMS, could be translated, as proposed in [113, 136], in the translations:  $\{+R(a, b, val_i)\}$ , for each possible value of  $val_i$ , and  $\{+Q(a, val_i), +R(a, val_i, val_j)\}$  for each possible value of  $val_j$  and each possible value of  $val_i \neq b$ . However, this seems us to be a “bigger change” to the database that is not strictly necessary for performing the desired update. The solution, proposed in this chapter, retrieves in this case, the unique translation  $\{+R(a, b, \perp)\}$ . The existential variable  $Z$  is fixed to the value  $\perp$  (the NULL value), that suffices, in the absence of any additional information specifying  $Z$ , to perform the desired view update. Intuitively,  $Q(a, b)$ , thought of as a ‘trustable’ fact, as it belongs to the extensional database, is used to ‘justify’ the construction of the translation.  $\square$

This chapter is a contribution to support view updating, consisting of insertion and deletion operations in deductive database, preventing the anomalies previous approaches suffer from. In fact, as shown before, existing approaches satisfy the update request by generating as many translations as the different values that can be assigned to the existential variables, whereas, intuitively, the approach proposed in this chapter, that could be defined *cautiously liberal*, limits the wide range of translations to those that are “supported” or validated by the deductive database.

*Example 5.2.* Consider the deductive database, obtained by extending the Example 5.1:

<sup>1</sup> An existential derived predicate is defined by a deductive rule containing variables in the body that do not occur in the head of the rule. Note that this situation is likely to occur in many real cases, e.g the simple case of a database view defined as a projection of a base relation.

<sup>2</sup> A flat integrity constraint is defined only in terms of base predicates, i.e. its definition does not contain view.

$$\begin{array}{ll} Q(a, b). & P(X) \leftarrow Q(X, Y), R(X, Y, Z) \\ S(a, b, c). & R(X, Y, Z) \leftarrow S(X, Y, Z), T(X, Y, Z) \end{array}$$

and the update request  $+P(a)$ . In this case the unique repair is  $\mathcal{R} = \{+T(a, b, c)\}$ . The proposed strategy implements a process that takes advantage of the initial knowledge, i.e. the set of extensional facts and the set of intensional facts derived through views of the deductive database. In this specific case, it recognizes that due to the insertion of  $T(a, b, c)$  the intensional fact  $R(a, b, c)$  can be derived and, consequently, the update request asking for the insertion of the fact  $P(a)$  can be justified.  $\square$

Previous discussion of still very simple cases introduces the serious conceptual problem underlying view update and justifies the flurry of research addressing this topic [14, 15, 21, 33, 43, 45, 55–57, 62, 77, 78, 85, 112, 113, 133, 136, 137]. However, the majority of these proposals work for restricted kinds of constraints, i.e flat-integrity constraints and in addition do not allow existential derived predicates. A detailed comparison with the few approaches, facing the same problem will be provided in Section 5.5.

### 5.1.1 Contribution

The novelty of the framework proposed here, consists in the definition of a formal declarative semantics for view updating that allows the identification, among the set of all possible repairs, of the subset of *supported repairs*, i.e. the repairs whose actions are “supported” by the database or by other updates. Given a deductive database and an update request, the computation of supported repairs is performed by rewriting the update request and the deductive database in the form of active integrity constraints. This chapter, that proposes a declarative semantics for view updating in the presence of existentially derived predicates and non-flat integrity constraints, avoids the anomaly previous approaches suffer from, that is the generation of as many translations as the different values that can be assigned to the existential variables. Specifically, the proposed repair semantics, to the best of our knowledge, considers, systematically and for the first time, the possible introduction of null values in the form they are present and treated in commercial DBMS: null values of the same type are used to restore consistency in the presence of desired view update if no additional (supported) information is available. Intuitively, the pragmatic solution encapsulated in this strategy limits the wide range of repairs, to those that are validated by the deductive database and recommends our approach be effectively implemented in commercial DBMS. Finally, the chapter proves the soundness and completeness of the proposal and presents some results on the complexity of computing supported repairs.

### 5.1.2 Plan of the Chapter

The remainder of this chapter is organized as follows. Section 5.2 formally introduces the problem of view updating and formalizes the proposal of a declarative semantics for view updating in deductive databases. Section 5.3 shows how “supported” repairs can be computed by rewriting the deductive database and the update request into active integrity constraints. Section 5.4 provides results on the complexity of computing supported repairs and introduces the main results of soundness and completeness. Section 5.5 surveys the related works, and provides some comparisons with the proposed approach.

## 5.2 A Declarative Semantics for View Updating

We briefly review the basic concept of deductive database [1, 105].

**Definition 5.3.** A *deductive database*  $\mathcal{J}$  is a tuple  $\langle \mathcal{I}, \mathcal{P}, \eta \rangle$ , where  $\mathcal{I}$  is a database,  $\mathcal{P}$  is a locally stratified logic program representing a set of views and  $\eta$  a *set of integrity constraints*.  $\square$

Given the deductive database  $\mathcal{J} = \langle \mathcal{I}, \mathcal{P}, \eta \rangle$ ,  $\mathcal{P}_{\mathcal{I}}$  is the unique stable model of  $\mathcal{P} \cup \mathcal{I}$ <sup>3</sup>. It represents the *knowledge* stored by  $\mathcal{J}$ , that is the facts belonging to  $\mathcal{I}$  and those derived through  $\mathcal{P}$ .  $\mathcal{J}$  is *consistent* if  $\mathcal{P}_{\mathcal{I}} \models \eta$ , that is if all integrity constraints in  $\eta$  are satisfied by  $\mathcal{P}_{\mathcal{I}}$ , otherwise it is *inconsistent*.

Predicate symbols that occur in the head of a view in  $\mathcal{P}$  are called *derived predicates* and are denoted as  $DPred(\mathcal{J})$ . Predicate symbols that appear in  $\mathcal{I}$  are *base predicates* and are denoted as  $BPred(\mathcal{J})$ . Moreover,  $Pred(\mathcal{J}) = BPred(\mathcal{J}) \cup DPred(\mathcal{J})$ . A base fact is also called *EDB fact*, and a derived fact is also called *IDB fact*. Given a set of update actions  $\mathcal{U}$  and a deductive database  $\mathcal{J}$ , we define the sets  $\mathcal{U}_{EDB} = \{\pm a(t) \mid \pm a(t) \in \mathcal{U} \wedge a \in BPred(\mathcal{J})\}$ ,  $\mathcal{U}_{IDB} = \{\pm a(t) \mid \pm a(t) \in \mathcal{U} \wedge a \in DPred(\mathcal{J})\}$ .

In order to allow the expression of each possible condition, the proposed framework allows the management of non-flat integrity constraints (that is integrity constraints also defined over derived predicates).

Now we formally introduces the problem of view updating and formalizes the proposal of a declarative semantics for insertion and deletion operations in deductive databases<sup>4</sup>.

**Definition 5.4.** REQUEST SET. A *request set*  $\mathcal{S}$  is a set of ground literals. Moreover,  $\mathcal{S}^{in} = \{a(t) \mid a(t) \in \mathcal{S}\}$  and  $\mathcal{S}^{out} = \{a(t) \mid not\ a(t) \in \mathcal{S}\}$ .  $\square$

<sup>3</sup> Observe that as  $\mathcal{P}_{\mathcal{I}}$  is locally stratified, it admits just one stable models

<sup>4</sup> The replacement of a fact is not directly managed. Obviously, it can be obtained by applying first a delete request and then an insert request.



Therefore,  $S^{in}$  denotes the set of atoms that is requested to be *true*; whereas  $S^{out}$  denotes the set of atoms that is requested to be *false*. A request set  $\mathcal{S}$  against a deductive database  $\mathcal{J}$  is, intuitively, accomplished by performing a minimal set of insert and delete operations of the stored base facts and in all the consequent update operations of the derived predicates.

**Definition 5.5.** KNOWLEDGE UPDATE. Given a deductive database  $\mathcal{J}$ , a *knowledge update*  $\mathcal{U}$  is a consistent set of ground update atoms such that  $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} = \mathcal{P}_{\mathcal{I} \circ \mathcal{U}_{EDB}}$   $\square$

Previous definition states that updates have to be consistently derived starting from updates over the extensional part of the deductive database. In other words the effect of applying a set of ground update atoms both extensional and intensional,  $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U}$ , is consistent if it can be simulated by just considering its extensional portion,  $\mathcal{U}_{EDB}$ , and then deducing all the consequences over  $\mathcal{I}$ ,  $\mathcal{P}_{\mathcal{I} \circ \mathcal{U}_{EDB}}$ .

**Definition 5.6.** KNOWLEDGE REPAIR. Given a deductive database  $\mathcal{J}$  and a request set  $\mathcal{S}$ , a knowledge update  $\mathcal{U}$  for  $\mathcal{J}$  is a *knowledge repair* with respect to  $\mathcal{S}$  if:

1. *it guarantees consistency*:  $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \models \eta$ ;
2. *it confirms the request set*:
  - $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \cap S^{in} = S^{in}$ ;
  - $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \cap S^{out} = \emptyset$ ;
3. *it is minimal*: there is no knowledge update  $\mathcal{W} \subset \mathcal{U}$  such that  $\mathcal{W}$  guarantees consistency and confirms the request set.

The set of knowledge repairs for  $\mathcal{J}$  with respect to  $\mathcal{S}$  is denoted as  $\mathbf{KR}(\mathcal{J}, \mathcal{S})$ .  $\square$

**Definition 5.7.** REPAIR. Let  $\mathcal{U}$  be a knowledge repair for a deductive database  $\mathcal{J}$  with respect to a request set  $\mathcal{S}$ . The set  $\mathcal{U}_{EDB}$  is a *repair* for  $\mathcal{J}$  with respect to  $\mathcal{S}$ . The set of repairs for  $\mathcal{J}$  with respect to  $\mathcal{S}$  is denoted as  $\mathbf{R}(\mathcal{J}, \mathcal{S})$ .  $\square$

Previous definition retrieves all the repairs or translations that satisfy the update request, ensuring minimality and constraints satisfaction. Therefore, the translations provided by the approaches in [113, 136] for the Example 5.1 are repairs. However, as also stated in introduction we do not consider satisfactory this solution as in the presence of existentially derived predicates the knowledge repair may contain insert operations not supported by the database. In order to provide the definition of supported atom, that intuitively states for an atom validated by the initial knowledge or by other supported update atoms, we refer to a knowledge repair  $\mathcal{U}$  and to  $\mathcal{U}^S \subseteq \mathcal{U}^+$ , where  $\mathcal{U}^S$  states for the set of update atoms for which a certificate of quality has already been provided, i.e. the subset of known supported atoms<sup>5</sup>.

<sup>5</sup> The recursive construction of this set will be formally provided in Definition 5.9.

**Definition 5.8.** SUPPORTED ATOM. Let  $\mathcal{J} = \langle \mathcal{I}, \mathcal{P}, \eta \rangle$  be a deductive database and  $\mathcal{U}^S$  a set of facts. Let  $a(t_X)$  be a ground instance of an atom  $a(X)$  and  $\gamma(t_Y)$  a ground instance of a conjunction of literals  $\gamma(Y)$ . We denote with  $\Lambda = X \cap Y$  the *common variables*, i.e. variables appearing in  $a$  and  $\gamma$  and with  $\Delta = X - Y$  the *free variables* of  $a$ .  $\langle a(t_\Lambda, t_\Delta), a(\Lambda, \Delta) \rangle$  is *supported* by  $\langle \gamma(t_Y), \gamma(Y) \rangle$  with respect to  $\langle \mathcal{J}, \mathcal{U}^S \rangle$  if:

1. *Free variables are instantiated to null:*  
 $t_\Delta = \perp$  and
2. *Common variables are instantiated by the initial knowledge or by atoms in  $\mathcal{U}^S$ :*  
 $\gamma(Y)$  contains a conjunction  $\eta(\Lambda, Z)$  of positive literals s.t. the corresponding ground conjunction in  $\gamma(t_Y)$  is in the form  $\eta(t_\Lambda, t_Z)$  and for each  $A$  in  $\eta(t_\Lambda, t_Z)$ 
  - $A \in \mathcal{P}_{\mathcal{DB}}$  or
  - $A \in \mathcal{U}^S$ . □

Therefore, intuitively, a *supported knowledge repair* is a knowledge repair in which each insert update atom is supported (i.e.  $\mathcal{U}^+ = \mathcal{U}^S$ ). A formal definition of this concept requires an in depth analysis of the structure of a deductive database.

**Definition 5.9.** SUPPORTED KNOWLEDGE REPAIR. Given a deductive database  $\mathcal{J} = \langle \mathcal{I}, \mathcal{P}, \eta \rangle$  and a request set  $\mathcal{S}$ , a knowledge repair  $\mathcal{U}$  is a *supported knowledge repair* if  $\mathcal{U}^+ = \mathcal{U}^S$ , where  $\mathcal{U}^S$  is defined as follows<sup>6</sup>: a fact  $a(t_X) \in \mathcal{U}^S$  iff  $a(t_X) \in \mathcal{U}^+$  and  $a(t_X)$  satisfies at least one of the following conditions:

1.  $a(t_X) \in \mathcal{S}^{in}$
2.  $\exists r \in \mathcal{P}$  of the form  $a(X) \leftarrow \beta(Y)$  and  
 $\exists g \in \text{ground}(r)$  of the form  $a(t_X) \leftarrow \beta(t_Y)$  such that
  - $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \models \beta(t_Y)$  and
  - $\langle a(t_X), a(X) \rangle$  is supported by  $\langle \beta(t_Y), \beta(Y) \rangle$  with respect to  $\langle \mathcal{J}, \mathcal{U}^S \rangle$
3.  $\exists r \in \mathcal{P}$  of the form  $b(H) \leftarrow a(X), \beta(Y)$  and  
 $\exists g \in \text{ground}(r)$  of the form  $b(t_H) \leftarrow a(t_X), \beta(t_Y)$  such that
  - $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \models \beta(t_Y)$  and
  - $b(t_H) \in \mathcal{U}^S$  and
  - $\langle a(t_X), a(X) \rangle$  is supported by  $\langle b(t_H) \wedge \beta(t_Y), b(H) \wedge \beta(Y) \rangle$  with respect to  $\langle \mathcal{J}, \mathcal{U}^S \rangle$
4.  $\exists r \in \mathcal{P}$  of the form  $b(H) \leftarrow \text{not } a(X), \beta(Y)$  and  
 $\exists g \in \text{ground}(r)$  of the form  $b(t_H) \leftarrow \text{not } a(t_X), \beta(t_Y)$  such that
  - $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \models \beta(t_Y)$  and
  - $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \not\models b(t_H)$  and
  - $\langle a(t_X), a(X) \rangle$  is supported by  $\langle \beta(t_Y), \beta(Y) \rangle$  with respect to  $\langle \mathcal{J}, \mathcal{U}^S \rangle$

<sup>6</sup> Observe that the definition of  $\mathcal{U}^S$  is recursive.

5.  $\exists r \in \eta$  of the form  $\text{not } a(X), \beta(Y) \supset$  and  
 $\exists g \in \text{ground}(r)$  of the form  $\text{not } a(t_X), \beta(t_Y) \supset$  such that
- $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \models \beta(t_Y)$  and
  - $\langle a(t_X), a(X) \rangle$  is supported by  $\langle \beta(t_Y), \beta(Y) \rangle$  with respect to  $\langle \mathcal{J}, \mathcal{U}^S \rangle$ .

The set of supported knowledge repairs for  $\mathcal{J}$  with respect to  $\mathcal{S}$  is denoted as  $\mathbf{JKR}(\mathcal{J}, \mathcal{S})$ .  $\square$

*Example 5.10.* Consider the Example 5.2 and the knowledge repair  $\mathcal{U} = \{+P(a), +T(a, b, c), +R(a, b, c)\}$ . The set  $\mathcal{U}^S$  is recursively constructed as follows.  $P(a) \in \mathcal{U}^S$  as it belongs to  $S^{in}$  (item 1).  $R(a, b, c) \in \mathcal{U}^S$  as there is a ground instance  $R(a, b, c) \leftarrow S(a, b, c), T(a, b, c)$  s.t.  $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \models S(a, b, c), T(a, b, c)$  and  $\langle R(a, b, c), R(X, Y, Z) \rangle$  is supported by  $\langle S(a, b, c) \wedge T(a, b, c), S(X, Y, Z) \wedge T(X, Y, Z) \rangle$  as  $S(a, b, c) \in \mathcal{P}_{\mathcal{DB}}$  (item 2). Finally, the same ground instance is used to validate the justification of  $T(a, b, c)$  (item 3). Therefore, by recursively applying Definition 5.9 we obtain  $\mathcal{U}^S = \{P(a), R(a, b, c), T(a, b, c)\}$ . As  $\mathcal{U}^S = \mathcal{U}^+$ , it follows that  $\mathcal{U}$  is a supported knowledge repair. Suppose now we add the integrity constraint  $T(X, Y, Z), \text{not } V(X, Y) \supset$ . In this case, in order to guarantee integrity constraint maintenance the update operation  $+V(a, b)$  has to be performed, and  $\mathcal{U}$  has to be extended by adding  $+V(a, b)$ .  $V(a, b) \in \mathcal{U}^S$  as there is a ground instance  $T(a, b, c), \text{not } V(a, b) \supset$  s.t.  $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \models T(a, b, c)$  and  $\langle V(a, b), V(X, Y) \rangle$  is supported by  $\langle T(a, b, c), T(X, Y, Z) \rangle$  as  $T(a, b, c) \in \mathcal{U}^S$  (item 5). Thus,  $\mathcal{U} = \{+P(a), +R(a, b, c), +T(a, b, c), +V(a, b)\}$  is a supported knowledge repair. The knowledge repair  $\mathcal{W} = \{+P(a), +Q(a, val_i), +R(a, val_i, val_j), +S(a, val_i, val_j), +T(a, val_i, val_j)\}$ , where  $val_i$  and  $val_j$  are generic constants, is not a supported knowledge repair.  $\square$

**Definition 5.11.** SUPPORTED REPAIR. Let  $\mathcal{U}$  be a supported knowledge repair for a deductive database  $\mathcal{J}$  with respect to a request set  $\mathcal{S}$ . The set  $\mathcal{U}_{EDB}$  is a *supported repair* for  $\mathcal{J}$  with respect to  $\mathcal{S}$ . The set of supported repairs for  $\mathcal{J}$  with respect to  $\mathcal{S}$  is denoted as  $\mathbf{JR}(\mathcal{J}, \mathcal{S})$ .  $\square$

Therefore, the concept of supported repair refines the concept of repair. In fact, note that the translations provided by the approaches in [113, 136] for the Example 5.1 albeit are repairs, are not supported repairs.

Given the deductive databases  $\mathcal{J} = \langle \mathcal{I}, \mathcal{P}, \eta \rangle$  and  $\mathcal{J}' = \langle \mathcal{I}, \mathcal{P}', \eta' \rangle$ ,  $\mathcal{J}'$  is a *standard version* of  $\mathcal{J}$  if:

- $\mathcal{P}'$  is obtained by rewriting each  $v \in \mathcal{P}$  into a set of *standard views* of the following four types:<sup>7</sup>

<sup>7</sup> A first-order query can be expressed, without loss of generality, by using selection, projection, join and negation [1]. As an example the deductive rule reported in Example 5.1 can be rewritten into:  $P'(X, Y, Z) \leftarrow Q(X, Y), R(X, Y, Z)$  and  $P(X) \leftarrow P'(X, Y, Z)$ .

PROJECTION VIEW :	$a(X) \leftarrow b(X, Y)$
NEGATION VIEW :	$a(X) \leftarrow b(X), \text{not } c(X)$
JOIN VIEW :	$a(X, Y, Z) \leftarrow b(X, Y), c(Y, Z)$
SELECTION VIEW :	$a(X) \leftarrow b(X), \phi(X)$

where  $X, Y$  and  $Z$  are lists of variables or constants and  $\phi$  is a built-in predicate used to evaluate a condition over  $X$ .

- $\eta'$  is obtained, by rewriting each  $ic \in \eta$  into a *standard integrity constraint* of the form  $a(X) \supset$  and a set of standard views.<sup>8</sup>

**Proposition 5.12.** Given the deductive databases  $\mathcal{J}$  and  $\mathcal{J}'$ , where  $\mathcal{J}'$  is a standard version of  $\mathcal{J}$ , and a request set  $\mathcal{S}$ , for each  $\mathcal{U}' \in \mathbf{JKR}(\mathcal{J}', \mathcal{S})$ ,  $\mathcal{U} \in \mathbf{JKR}(\mathcal{J}, \mathcal{S})$ , where  $\mathcal{U} = \mathcal{U}' - \{\pm a(t) \mid \pm a(t) \in \mathcal{U}' \wedge a \notin \text{Pred}(\mathcal{J})\}$ .

**Proof sketch.**

It is possible to prove the proposition showing that by collapsing two generic views of the standard deductive database  $\mathcal{J}' = \langle \mathcal{I}, \mathcal{P}', \eta' \rangle$  we obtain a (non standard) deductive database  $\mathcal{J}'' = \langle \mathcal{I}, \mathcal{P}'', \eta'' \rangle$  such that the set  $\mathcal{U}'' = \mathcal{U}' - \{\pm a(t) \mid \pm a(t) \in \mathcal{U}' \wedge a \notin \text{Pred}(\mathcal{J}'')\}$  is a supported knowledge repair for  $\mathcal{J}''$  with respect to  $\mathcal{S}$ . Iteratively, an analogous reasoning can be applied, so that finally obtaining  $\mathcal{J}$  and the corresponding set of update atoms  $\mathcal{U}$  s.t.  $\mathcal{U}^+$  is supported.  $\square$

As a consequence of the above result, in the rest of this chapter we just consider deductive databases in standard version.

### 5.3 Rewriting into Active Integrity Constraints

The tool for performing view updating based on the semantics presented in Section 5.2, consists in the transformation of a deductive database and a request set into a set of AICs. The basic idea is to extract from a deductive database  $\mathcal{J}$  and a request set  $\mathcal{S}$ , a set of AICs whose purpose is to react to the updates underlying  $\mathcal{S}$ , by modifying  $\mathcal{I}$  in order to justify the updated knowledge. More formally, our goal is to find a set of AICs,  $\text{Rew}(\mathcal{J}, \mathcal{S})$ , such that the *supported knowledge repairs of  $\mathcal{J}$  with respect to  $\mathcal{S}$* ,  $\mathbf{JKR}(\mathcal{J}, \mathcal{S})$ , can be derived from the set of *founded repairs of the original knowledge  $\mathcal{P}_{\mathcal{DB}}$  with respect to these active integrity constraints*,  $\mathbf{FR}(\mathcal{P}_{\mathcal{I}}, \text{Rew}(\mathcal{J}, \mathcal{S}))$ .

In the following, we will use the special constant ‘\_’ (*placeholder*). Intuitively, it states for a generic value, i.e. a constant value or the NULL value ( $\perp$ ). Given the lists of terms (variables or constants)  $X = X_1, \dots, X_n$  and  $Y = Y_1, \dots, Y_n$ ,

1.  $X \neq \_$  is shorthand for  $\bigwedge_{i=1}^n (X_i \neq \_)$

<sup>8</sup> As an example the foreign key constraint  $R(X), (\exists Y) Q(X, Y) \supset$  can be rewritten into the standard constraint  $V(X) \supset$  the projection view  $P(X) \leftarrow Q(X, Y)$  and the negation view  $V(X) \leftarrow R(X), \text{not } P(X)$ .

2.  $X \preceq Y$  ( $X$  is not more defined than  $Y$ ) is shorthand for  $\bigwedge_{i=1}^n ((X_i = Y_i) \vee (X_i = \_ \wedge Y_i \text{ IS NOT NULL}) \vee (X_i \text{ IS NULL} \wedge Y_i \text{ IS NULL}))$
3.  $X \prec Y$  ( $X$  is less defined than  $Y$ ) is shorthand for  $X \preceq Y \wedge Y \neq \_$

As an example,  $(2, \_, \_ \perp) \preceq (2, 3, \_ \perp)$ , whereas  $(2, 3, \_ \perp) \prec (2, 3, \perp, \perp)$ . We say that a fact  $p(x)$  is *fully-defined* if  $x \neq \_$ , and *partially-defined* otherwise.

Given a deductive database  $\mathcal{J}$  and a set of update atoms  $\mathcal{FR}$ ,  $FullyDefined(\mathcal{FR}) = \{\pm a(t) \mid \pm a(t) \in \mathcal{U} \wedge a \in Pred(\mathcal{J}) \wedge a(t) \text{ is fully-defined}\}$ .

The rest of this section reports the rewriting into active integrity constraints of a request set  $\mathcal{S}$  and a deductive database  $\mathcal{J} = \langle \mathcal{I}, \mathcal{P}, \eta \rangle$ .

**Definition 5.13.** REWRITING OF  $\mathcal{S}$ . Given a request set  $\mathcal{S}$ ,  $Rew(\mathcal{S})$  is the set of the AICs:

$$\begin{aligned} \mathbf{Req}_1 &: \text{not } a(x) \supset +a(x) && \text{for each } a(x) \in \mathcal{S}^{in} \\ \mathbf{Req}_2 &: a(x) \supset -a(x) && \text{for each } a(x) \in \mathcal{S}^{out} \quad \square \end{aligned}$$

A set of auxiliary active integrity constraints is needed in order to ensure the computation of supported knowledge repairs.

**Definition 5.14.** AUXILIARY AICs. Given a deductive database  $\mathcal{J}$ ,  $Aux(\mathcal{J})$  is the set of the AICs:

$$\begin{aligned} \mathbf{Aux}_1 &: a(X), a(X'), X \prec X', \text{not } fixed^a(X) \supset +fixed^a(X) \\ &\quad \text{for each predicate } a \text{ of } \mathcal{J} \text{ and} \\ \mathbf{Aux}_2 &: a(X, \_), \text{not } fixed^a(X, \_), \text{not } a(X, \perp) \supset +a(X, \perp) \\ &\quad \text{if } a \text{ is a base predicate or} \\ \mathbf{Aux}_3 &: a(X), \text{not } fixed^a(X) \supset \\ &\quad \text{if } a \text{ is a derived predicate.} \quad \square \end{aligned}$$

Each AIC in the set  $\mathbf{Aux}_1$  states that a partially-defined fact is *fixed* if it is supported by a fully-defined fact. The AICs  $\mathbf{Aux}_2$  replaces each ‘ $\_$ ’ occurring in EDB facts with  $\perp$ . Finally the AICs in  $\mathbf{Aux}_3$  (observe that these are simple integrity constraints as their heads are empty) assert that each IDB fact must be fixed, otherwise no founded repair is computed. The reason for these auxiliary rules, will be made clear considering the generating mechanism encapsulated in the rewriting process. Anyhow, it should be stressed that they are necessary to guarantee the computation of the reliable repairs, i.e. the supported knowledge repairs only. Intuitively, the set of AICs, obtained from the rewriting, propagate the placeholder ‘ $\_$ ’, stating for a generic value during the attempt to compute a founded repair, with the final aim of having it instantiated to a fixed value. Whenever this is not the case and we are in the presence of a not fixed IDB fact, then the attempt to generate a founded repair fails<sup>9</sup>; on the other hand ‘ $\_$ ’ is converted into  $\perp$  in the case of an EDB fact. Note that

<sup>9</sup> This guarantees that only supported repairs are computed so avoiding the problem of generating a wide range of repairs one for each possible instantiation.

this pragmatic solution recommends our approach for implementation in commercial DBMS.

In the following, we provide the rewriting of  $\mathcal{P}$ . For the sake of simplicity, we assume that each derived predicate is defined by a view. However, a simple extension allows to handle the union operator.

**Definition 5.15.** REWRITING OF PROJECTION VIEWS. *Given a projection view  $v$  of the form  $a(X) \leftarrow b(X, Y)$ ,  $Rew(v)$  is the set of the AICs:*

$$\begin{aligned} \mathbf{r}_1 &: \text{not } a(X), b(X, Y), X \neq \_ \supset +a(X) \vee -b(X, Y) \\ \mathbf{r}_2 &: b(X, Y), X \neq \_, \text{not supported}^a(X) \supset +\text{supported}^a(X) \\ \mathbf{r}_3 &: a(X), X \neq \_, \text{not supported}^a(X) \supset -a(X) \\ \mathbf{r}_4 &: a(X), \text{not } b(X, \_) \supset +b(X, \_) \quad \square \end{aligned}$$

The AIC  $\mathbf{r}_1$  states that if the body of the view,  $b(X, Y)$ , is *true*, and its head,  $a(X)$ , is *false* then, in order to guarantee the consistency of the updated knowledge, either the action of inserting  $a(X)$  or the action of deleting  $b(X, Y)$  has to be performed. The AIC  $\mathbf{r}_2$  states that if the updated knowledge contains a fact  $b(X, Y)$ , with  $X \neq \_$ , then the fact  $a(X)$  is supported ( $\text{supported}^a(X)$  is *true*), while the AIC  $\mathbf{r}_3$  ensures that the updated knowledge does not contain any *unsupported* fact  $a(X)$ . Finally, the meaning of  $\mathbf{r}_4$  is that if the updated knowledge contains  $a(X)$ , then it must contain the fact  $b(X, \_)$ . Note that, the auxiliary AICs, previously presented, infer from  $b(X, \_)$  the unique fact  $b(X, \perp)$  if  $b$  is a base predicate and no other fact  $b(X, Y)$ , with  $Y \neq \perp$ , is inferred in the updated knowledge; whereas they reject the repair if  $b$  is a derived predicate and  $b(X, \_)$  is not fixed by a fully-defined fact.

*Example 5.16.* Consider the deductive database  $\mathcal{J} = \langle \emptyset, \{a(X) \leftarrow b(X, Y)\}, \emptyset \rangle$ , and the request set  $\mathcal{S} = \{a(1)\}$ . Consider the set of AICs constituted by  $Rew(\mathcal{S}) = \{\text{not } a(1) \supset +a(1)\}$ ,  $Rew(\mathcal{P})$  (consisting of  $\mathbf{r}_1$ ,  $\mathbf{r}_2$  and  $\mathbf{r}_3$  defined as above) and the set of auxiliary AICs in  $Aux(\mathcal{J})$ , reported in the following:

$$\begin{aligned} &b(X, Y), b(X', Y'), XY \prec X'Y', \text{not fixed}^b(X, Y) \supset +\text{fixed}^b(X, Y) \\ &b(X, Y), \text{not fixed}^b(X, Y), Y = \_, \text{not } b(X, \perp) \supset +b(X, \perp) \\ &a(X), a(X'), X \prec X', \text{not fixed}^a(X) \supset +\text{fixed}^a(X) \\ &a(X), \text{not fixed}^a(X) \supset \end{aligned}$$

The unique founded repair for  $\mathcal{P}_{\mathcal{I}}$  with respect to this set of AICs is  $\mathcal{FR} = \{+a(1), +b(1, \_), +b(1, \perp), +\text{supported}^a(1), +\text{fixed}^a(1), +\text{fixed}^b(1, \perp)\}$  corresponding to the unique supported knowledge repair for  $\mathcal{J}$  with respect to  $\mathcal{S}$ :  $FullyDefined(\mathcal{FR}) = \{+a(1), +b(1, \perp)\}$  and to the supported repair  $FullyDefined(\mathcal{FR})_{EDB} = \{+b(1, \perp)\}$ . Consider now the deductive database  $\mathcal{J} = \langle \{b(1, \perp)\}, \{a(X) \leftarrow b(X, Y)\}, \emptyset \rangle$  and the request set  $\mathcal{S} = \{\text{not } b(1, \perp)\}$ . The new founded repair is  $\mathcal{FR} = \{-b(1, \perp), -a(1)\}$ .  $\square$

**Definition 5.17.** REWRITING OF NEGATION VIEWS.

Given a negation view  $v$  of the form  $a(X) \leftarrow b(X), \text{not } c(X)$ ,  $\text{Rew}(v)$  is the set of the AICs:

- $\mathbf{r}_1$  :  $\text{not } a(X), b(X), \text{not } c(X), X \neq - \supset +a(X) \vee -b(X) \vee +c(X)$
- $\mathbf{r}_2$  :  $b(X), \text{not } c(X), X \neq -, \text{not supported}^a(X) \supset +\text{supported}^a(X)$
- $\mathbf{r}_3$  :  $a(X), X \neq -, \text{not supported}^a(X) \supset -a(X)$
- $\mathbf{r}_4$  :  $a(X), b(X'), X \prec X', \text{not supportable}_{-c}^a(X, X') \supset +\text{supportable}_{-c}^a(X, X')$ ,
- $\mathbf{r}_5$  :  $\text{supportable}_{-c}^a(X, X'), \text{supported}^a(\bar{X}), X \prec \bar{X}, X' \neq \bar{X},$   
 $\text{not unnecessary}_{-c}^a(X, X') \supset +\text{unnecessary}_{-c}^a(X, X')$
- $\mathbf{r}_6$  :  $\text{supportable}_{-c}^a(X, X'), \text{not unnecessary}_{-c}^a(X, X'), c(X') \supset -c(X')$
- $\mathbf{r}_7$  :  $a(X), X \neq -, \text{not supportable}_{+b, -c}^a(X) \supset +\text{supportable}_{+b, -c}^a(X)$
- $\mathbf{r}_8$  :  $\text{supportable}_{+b, -c}^a(X), \text{not } b(X) \supset +b(X)$
- $\mathbf{r}_9$  :  $\text{supportable}_{+b, -c}^a(X), c(X) \supset -c(X)$  □

The meaning of the AICs  $\mathbf{r}_1$ ,  $\mathbf{r}_2$  and  $\mathbf{r}_3$  is similar to the one discussed for projection views. The AIC  $\mathbf{r}_4$  states that if  $a(X)$  is present in the updated knowledge and exists an atom  $b(X')$ , with  $X \prec X'$ , then  $a(X)$  can be supported by (eventually) deleting  $c(X')$  ( $\text{supportable}_{-c}^a(X, X')$  is true). The AIC  $\mathbf{r}_5$  states that if it is possible to support  $a(X)$  by acting on  $c(X')$ , but  $a(X)$  is fixed by another atom  $a(\bar{X})$ , already supported, is unnecessary to support  $a(X)$  by acting on  $c(X')$  ( $\text{unnecessary}_{-c}^a(X, X')$  is true). The AIC  $\mathbf{r}_6$  states that we can support  $a(X)$  through  $c(X')$  if this is possible and not unnecessary, whereas  $\mathbf{r}_7$  states that if  $a(X)$  belongs to the updated knowledge and it is fully-defined, it must be supported by ensuring the presence of  $b(X)$  and the absence of  $c(X)$ . The AICs  $\mathbf{r}_8$  and  $\mathbf{r}_9$  (eventually) insert  $b(X)$  and delete  $c(X)$ .

*Example 5.18.* Consider the deductive database  $\mathcal{J} = \langle \{b(1), c(1)\}, \{a(X) \leftarrow b(X), \text{not } c(X)\}, \emptyset \rangle$  and the request set  $\mathcal{S} = \{a(1)\}$ . Consider the set of AICs constituted by  $\text{Rew}(\mathcal{S})$ ,  $\text{Rew}(\mathcal{P})$  and  $\text{Aux}(\mathcal{J})$ . The unique founded repair of  $\mathcal{P}_{\mathcal{I}}$  with respect to this set of AICs is  $\mathcal{FR} = \{+a(1), +\text{supportable}_{-c}^a(1, 1), +\text{supportable}_{+b, -c}^a(1), -c(1), +\text{supported}^a(1), +\text{fixed}^a(1)\}$  corresponding to the supported knowledge repair  $\text{FullyDefined}(\mathcal{FR}) = \{+a(1), -c(1)\}$  and to the supported repair  $\text{FullyDefined}(\mathcal{FR})_{EDB} = \{-c(1)\}$ . Suppose now the request set is  $\mathcal{S} = \{d\}$  and  $\mathcal{P}$  also contains the projection rule  $d \leftarrow a(X)$ . Obviously, now the set of AICs also contains the AICs obtained from the rewriting of this projection rule (see Definition 5.15). The fact  $d$  is present in the knowledge base only if the relation  $a$  is not empty. The founded repair of  $\mathcal{P}_{\mathcal{I}}$  with respect to the new set of AICs is  $\mathcal{FR} = \{+d, +a(-), +\text{supported}^d, +\text{supportable}_{-c}^a(-, 1), -c(1), +a(1), +\text{supported}^a(1), +\text{supportable}_{-c}^a(1, 1), +\text{supportable}_{+b, -c}^a(1, 1), +\text{fixed}^a(-), +\text{fixed}^d, +\text{fixed}^a(1)\}$ . This founded repairs corresponds to the supported knowledge repair  $\text{FullyDefined}(\mathcal{FR}) = \{+d, +a(1), -c(1)\}$  and to the supported repair  $\text{FullyDefined}(\mathcal{FR})_{EDB} = \{-c(1)\}$ . □

**Definition 5.19.** REWRITING OF JOIN VIEWS. Given a join view  $v$  of the form  $a(X, Y, Z) \leftarrow b(X, Y), c(Y, Z)$ ,  $Rew(v)$  is the set the AICs:

- $\mathbf{r}_1$  :  $not\ a(X, Y, Z), b(X, Y), c(Y, Z), Y \neq - \supset$   
 $+ a(X, Y, Z) \vee -b(X, Y) \vee -c(Y, Z)$
- $\mathbf{r}_2$  :  $b(X, Y), c(Y, Z), Y \neq -, not\ supported^a(X, Y, Z) \supset$   
 $+ supported^a(X, Y, Z)$
- $\mathbf{r}_3$  :  $a(X, Y, Z), XYZ \neq -, not\ supported^a(X, Y, Z) \supset -a(X, Y, Z)$
- $\mathbf{r}_4$  :  $a(X, Y, Z), b(X', Y'), X \preceq X', Y \prec Y',$   
 $not\ supporttable_{+c}^a(X, Y, Z, X', Y') \supset +supporttable_{+c}^a(X, Y, Z, X', Y')$
- $\mathbf{r}_5$  :  $supporttable_{+c}^a(X, Y, Z, X', Y'), supported^a(\overline{X}, \overline{Y}, \overline{Z}),$   
 $XZ \preceq \overline{XZ}, Y \prec \overline{Y}, X'Y'Z \neq \overline{XYZ},$   
 $not\ unnecessary_{+c}^a(X, Y, Z, X', Y') \supset +unnecessary_{+c}^a(X, Y, Z, X', Y')$
- $\mathbf{r}_6$  :  $supporttable_{+c}^a(X, Y, Z, X', Y'), not\ unnecessary_{+c}^a(X, Y, Z, X', Y'),$   
 $not\ c(Z, Y') \supset +c(Z, Y')$
- $\mathbf{r}_7$  :  $a(X, Y, Z), c(Y', Z'), Z \preceq Z', Y \prec Y',$   
 $not\ supporttable_{+b}^a(X, Y, Z, Y', Z') \supset +supporttable_{+b}^a(X, Y, Z, Y', Z')$
- $\mathbf{r}_8$  :  $supporttable_{+b}^a(X, Y, Z, Y', Z'), supported^a(\overline{X}, \overline{Y}, \overline{Z}),$   
 $XZ \preceq \overline{XZ}, Y \prec \overline{Y}, XY'Z' \neq \overline{XYZ},$   
 $not\ unnecessary_{+b}^a(X, Y, Z, Y', Z') \supset +unnecessary_{+b}^a(X, Y, Z, Y', Z')$
- $\mathbf{r}_9$  :  $supporttable_{+b}^a(X, Y, Z, Y', Z'), not\ unnecessary_{+b}^a(X, Y, Z, Y', Z'),$   
 $not\ b(X, Y') \supset +b(X, Y')$
- $\mathbf{r}_{10}$  :  $a(X, Y, Z), Y \neq -, not\ supporttable_{+b,+c}^a(X, Y, Z) \supset$   
 $+ supporttable_{+b,+c}^a(X, Y, Z)$
- $\mathbf{r}_{11}$  :  $supporttable_{+b,+c}^a(X, Y, Z), not\ b(X, Y) \supset +b(X, Y)$
- $\mathbf{r}_{12}$  :  $supporttable_{+b,+c}^a(X, Y, Z), not\ c(Y, Z) \supset +c(Y, Z)$  □

The description of the intuitive meaning of the AICs, obtained from the rewriting of a join view, is left out as it is very similar to the one reported for negation views.

*Example 5.20.* Consider the deductive database  $\mathcal{J} = \langle \emptyset, \{a(X, Y, Z) \leftarrow b(X, Y), c(Y, Z)\}, \emptyset \rangle$ , the request set  $\mathcal{S} = \{a(1, 2, 3)\}$  and the set of AICs  $Rew(\mathcal{S})$ ,  $Rew(\mathcal{P})$  and  $Aux(\mathcal{J})$ . The unique founded repair for  $\mathcal{P}_{\mathcal{I}}$  with respect to the AICs is  $\mathcal{FR} = \{+a(1, 2, 3), +supporttable_{+c}^a(1, 2, 3, 1, 2), +supporttable_{+c}^a(1, 2, 3, 2, 3), +supporttable_{+b,+c}^a(1, 2, 3), +b(1, 2), +c(2, 3), +supported^a(1, 2, 3), +fixed^a(1, 2, 3), +fixed^b(1, 2), +c^{fixed}(2, 3)\}$  corresponding to the supported knowledge repair  $FullyDefined(\mathcal{FR}) = \{+a(1, 2, 3), +b(1, 2), +c(2, 3)\}$  and to the supported repair  $FullyDefined(\mathcal{FR})_{EDB} = \{+b(1, 2)\}$ . If  $\mathcal{P}$  also contains the view  $d(X, Z) \leftarrow a(X, Y, Z)$  and the request set is  $\mathcal{S} = \{d(1, 3)\}$ , no founded repair exist. Consider now  $\mathcal{J} = \langle \{b(1, 2), c(2, 3)\}, \{a(X, Y, Z) \leftarrow b(X, Y), c(Y, Z)\}, \emptyset \rangle$  and  $\mathcal{S} = \{not\ a(1, 2, 3)\}$ . In this case there are two founded repairs,  $\mathcal{FR}_1 = \{-a(1, 2, 3)\}$ ,



$-b(1, 2), +fixed^c(2, 3)$  and  $\mathcal{FR}_2 = \{-a(1, 2, 3), -c(2, 3), +fixed^b(1, 2)\}$ , that correspond respectively to the supported knowledge repairs  $FullyDefined(\mathcal{FR}_1) = \{-a(1, 2, 3), -b(1, 2)\}$  and  $FullyDefined(\mathcal{FR}_2) = \{-a(1, 2, 3), -c(2, 3)\}$  and to the supported repairs  $FullyDefined(\mathcal{FR}_1)_{EDB} = \{-b(1, 2)\}$  and  $FullyDefined(\mathcal{FR}_2)_{EDB} = \{-c(2, 3)\}$ .  $\square$

**Definition 5.21.** REWRITING OF SELECTION VIEWS. *Given a selection view  $v$  of the form  $a(X) \leftarrow b(X), \phi(X)$ ,  $Rew(v)$  is the set of the AICs*

- $\mathbf{r}_1$  :  $not\ a(X), b(X), \phi(X), X \neq - \supset +a(X) \vee -b(X)$
- $\mathbf{r}_2$  :  $b(X), \phi(X), X \neq -, not\ supported^a(X) \supset +supported^a(X)$
- $\mathbf{r}_3$  :  $a(X), X \neq -, not\ supported^a(X) \supset -a(X)$
- $\mathbf{r}_4$  :  $a(X), \phi(X), X \neq -, not\ b(X) \supset +b(X)$   $\square$

**Definition 5.22.** REWRITING OF  $\eta$ . *Given a constraint  $ic$  of the form  $a(X) \supset$ ,  $Rew(ic)$  denotes the set containing the AIC:  $a(X), X \neq - \supset -a(X)$ .  $\square$*

*Example 5.23.* Consider the request set  $\mathcal{S} = \{a(1, 2)\}$  and the deductive database  $\mathcal{J} = \langle \{a(1, 1)\}, \{b(X, Y_1, Y_2) \leftarrow a(X, Y_1), a(X, Y_2); c(X, Y_1, Y_2) \leftarrow b(X, Y_1, Y_2), Y_1 \neq Y_2\}, \{c(X, Y_1, Y_2) \supset\} \rangle$ . Observe that the views in  $\mathcal{P}$  and the integrity constraint in  $\eta$  express the key constraint  $a(X, Y_1), a(X, Y_2), Y_1 \neq Y_2 \supset$ . Intuitively, as the request set requires the presence of  $a(1, 2)$  conflicting with  $a(1, 1)$  the atom  $a(1, 1)$  has to be deleted. Indeed, the supported repair is  $\{+a(1, 2), -a(1, 1)\}$ .  $\square$

**Definition 5.24.** REWRITING OF  $\mathcal{J}$  AND  $\mathcal{S}$ . *Given a deductive database  $\mathcal{J} = \langle \mathcal{I}, \mathcal{P}, \eta \rangle$  and a request set  $\mathcal{S}$ , we denote as  $Rew(\mathcal{J}, \mathcal{S})$  the set of AICs  $Rew(\mathcal{S}) \cup Rew(\mathcal{P}) \cup Rew(\eta) \cup Aux(\mathcal{J})$  where  $Rew(\mathcal{P}) = \bigcup_{v \in \mathcal{LP}} Rew(v)$  and  $Rew(\eta) = \bigcup_{ic \in \mathcal{IC}} Rew(ic)$ .  $\square$*

**Fact 5.25** *Given a deductive database  $\mathcal{J} = \langle \mathcal{I}, \mathcal{P}, \eta \rangle$  and a request set  $\mathcal{S}$ , the complexity of constructing  $Rew(\mathcal{J}, \mathcal{S})$  is polynomial time.  $\square$*

## 5.4 Soundness, Completeness and Complexity Results

**Theorem 5.26.** (Soundness). *Let  $\mathcal{J} = \langle \mathcal{I}, \mathcal{P}, \eta \rangle$  be a deductive database and  $\mathcal{S}$  a request set. For every founded repair  $\mathcal{FR}$  for  $\langle \mathcal{P}_{\mathcal{I}}, Rew(\mathcal{J}, \mathcal{S}) \rangle$ ,  $FullyDefined(\mathcal{FR})$  is a supported knowledge repair for  $\mathcal{J}$  with respect to  $\mathcal{S}$ .*

**Proof sketch.** Let  $\mathcal{U} = FullyDefined(\mathcal{FR})$ . We show that 1)  $\mathcal{U}$  is a knowledge update, 2)  $\mathcal{U}$  is a knowledge repair and 3)  $\mathcal{U}$  is a supported knowledge repair.

1. To prove that  $\mathcal{U}$  is a knowledge update, suppose by contradiction that  $\exists a(t) \in \mathcal{P}_{\mathcal{I}} \circ \mathcal{U}$  s.t.  $\nexists g : a(t) \leftarrow \beta(y) \in ground(\mathcal{P}_{\mathcal{I}}) \mid \mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \not\models \beta(y)$ . The atom  $+supported^a(t)$  is not derived as the AIC  $\mathbf{r}_2$  of  $Rew(g)$  is not violated and, consequentially,  $\mathbf{r}_3$  is violated. So,  $\mathcal{FR}$  is not a repair.

2.  $\mathcal{U}$  is a knowledge repair because it *guarantees consistency*, i.e.  $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U} \models \eta$  as  $\mathcal{FR}(\mathcal{P}_{\mathcal{I}}) \models \text{Rew}(\eta)$ .  $\mathcal{U}$  confirms the request set as for each  $a(t) \in \mathcal{S}^{in}$ ,  $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U}$  contains  $a(t)$ . In fact, the AIC  $\text{not } a(t) \supset +a(t)$ , obtained by the rewriting of  $\mathcal{S}$ , is satisfied by  $\mathcal{FR}(\mathcal{P}_{\mathcal{I}})$  and by  $\mathcal{P}_{\mathcal{I}} \circ \mathcal{U}$ . Similarly, it can be shown that for each literal  $\text{not } a(t) \in \mathcal{S}^{out}$ ,  $a(t) \notin \mathcal{P}_{\mathcal{I}} \circ \mathcal{U}$ . Finally, *minimality* of  $\mathcal{U}$  is guaranteed by minimality of  $\mathcal{FR}$ .
3.  $\mathcal{U}$  is a supported knowledge repair as the structure of the AICs in  $\text{Rew}(\mathcal{J}, \mathcal{S})$  ensures that each atom in  $\mathcal{U}^+ \subset \mathcal{FR}^+$  is supported. In fact, the AICs in  $\text{Rew}(\mathcal{J}, \mathcal{S})$  that allow to infer insert update atoms are of two forms: (i)  $\text{not } a(t) \supset +a(t)$ ; (ii)  $\beta(X, Y), \text{not } a(X) \supset +a(X) \vee \Phi$ , where  $\Phi$  is eventually empty. When, an atom  $a(t)$  is inserted by the first AIC it is supported as it belongs to  $\mathcal{S}^{in}$ . An atom  $a(t_X)$  is inserted by the second AIC only if  $X$  is instantiated by  $\beta$ , i.e. only if  $\langle a(t_X), a(X) \rangle$  is supported by  $\langle \beta(t_Y), \beta(Y) \rangle$ .  $\square$

**Theorem 5.27.** (*Completeness*). *Let  $\mathcal{J} = \langle \mathcal{I}, \mathcal{P}, \eta \rangle$  be a deductive database and  $\mathcal{S}$  a request set. For every supported knowledge repair  $\mathcal{U}$  for  $\mathcal{J}$  with respect to  $\mathcal{S}$  there exists a founded repair  $\mathcal{FR}$  for  $\langle \mathcal{P}_{\mathcal{I}}, \text{Rew}(\mathcal{J}, \mathcal{S}) \rangle$  such that  $\mathcal{U} = \text{FullyDefined}(\mathcal{FR})$ .*

**Proof sketch.** This proof can be done by construction. It can be created a set  $\mathcal{FR}$  by adding to  $\mathcal{U}$  all insert update atoms of the form  $+a(t, -)$ ,  $+fixed^a(t)$ ,  $+supported^a(t)$ ,  $+supportable_{-c}^a(t)$ ,  $+unnecessary_{-c}^a(t)$ ,  $+supportable_{+b, -c}^a(t)$ ,  $+supportable_{+c}^a(t)$ ,  $+unnecessary_{+c}^a(t)$ ,  $+supportable_{+b, +c}^a(t)$  necessary to satisfy the AICs in  $\text{ground}(\text{Rew}(\mathcal{J}, \eta))$ . Obviously,  $\mathcal{FR}$  is a repair. Moreover, it can be shown that it is founded.  $\square$

**Theorem 5.28.** *Let  $\mathcal{J}$  be a deductive database and  $\mathcal{S}$  a request set, then the problem of checking (i) if there exists a supported knowledge repair  $\mathcal{U}$  for  $\mathcal{J}$  with respect to  $\mathcal{S}$  is  $\Sigma_2^p$ -complete; (ii) whether a ground atom  $g$  belongs to all repaired deductive databases obtained by means of supported knowledge repairs is  $\Pi_2^p$ -complete.*

**Proof sketch.** Both (i) and (ii) straightforward respectively from Theorem 5.28 and 3.33.  $\square$

## 5.5 Related Works

Over the years, a substantial amount of research has been devoted to the various issues surrounding view updating and not surprisingly a wide selection of approaches to the view update problem has evolved [14, 15, 21, 33, 43, 45, 55–57, 62, 77, 78, 85, 112, 113, 133, 136]. See [62, 112] for surveys of methods for view updating. In [77], the extremes are called *closed* and *open* update strategy. The first are very conservative and systematic [78] and are mainly based on the seminal work of Bancilhon and Spyrtatos in [15], whereas open strategies are very liberal and allow us to obtain as many solutions as possible.

View updating in definite deductive database, i.e. database where view predicates can only be defined by means of function free Horn rules and without negation, is investigated in [14]. In [57] an interesting model theoretic approach to view updates in deductive databases which encompasses a wide class of Herbrand semantics is proposed, including the perfect and stable model semantics for disjunctive databases with negation. In [85] the update of a single view is obtained by extending the relational model with identifiers on the values; whereas in [133] the view update problem is translated into a constraint satisfaction problem.

In the rest of this section we concentrate on the few works facing the view update problem, within the same dimension, i.e. considering deductive database, non-flat integrity constraints, the two basic update operations of insertion and deletion and that explicitly treat the case of existentially derived predicates [56, 113, 136]. As also stated in the introduction, our approach differs from the proposals in [113, 136] as in the presence of existential variables these techniques generate as many translations as the different values that can be assigned to them, whereas we only produce repairs supported by the deductive database. The alternative recent proposal of Ferré, Teniente and Urpi in [56], associates to an update request a set of *canonical translations* [140], each defined as a pair  $\langle T, C \rangle$ , where  $T$  is a set of base event facts, whose arguments may be either constants or skolem constants and  $C$  is a set of inequality constraints that skolem constants in  $T$  must satisfy. There are some similarities between the approach proposed here and the one in [56]: they both search for an effective update strategy in the presence of existential derived predicates and overcome the drawbacks of previously proposed methods avoiding the computation of all instantiations. However, as will be made clear in the rest of this section, they are significantly different in the case of existentially derived predicates. For the Example 5.1 the approach in [56] produces the two canonical translations:  $\langle \{+R(a, b, 0)\}, \{\emptyset\} \rangle$  and  $\langle \{+Q(a, 0), +R(a, 0, 1)\}, \{0 \neq b\} \rangle$ , whereas we obtain the unique translation  $\langle \{+R(a, b, \perp)\} \rangle$ . Therefore, in the absence of any additional information specifying an existential variable, the approach in [56] introduces skolem constants that propagate through rules and constraints and generate canonical translations containing *patterns* of the variable instantiations that are relevant for the update request, rather than taking into account all the possible instantiations; in any case, each canonical translation represents several extensional translations obtained by replacing skolem constants with values satisfying the inequality constraints. On the contrary, our approach adopts a solution similar to that of commercial DBMS, assigning to an existential variable, in the absence of any additional (supported) information, a unique null value.



## Conclusions

The main contributions of the thesis can be summarized as following:

Chapter 3 has introduced *active integrity constraints*, a simple and powerful form of active rules with declarative semantics, well suited for computing database repairs and consistent answers. The novelty of the approach proposed consists in the definition of a formal declarative semantics which allows us to identify, among the set of all possible repairs, the subset of *founded repairs* whose actions are specified in the head of rules and are *supported* by the database or by other updates. It has been shown that the computation of founded repairs can be done by rewriting the constraints into an (extended) Datalog program and computing the stable models of the program; the founded repairs are obtained by selecting, for each stable model, the set of “update actions”. We have also studied the properties of active integrity constraints and shown that for each production rule  $r$  update head atoms not making the conjunction of body literals *false* with respect to the repaired database (that is such that the body integrity constraint is satisfied), are useless. The thesis has also studied the computational complexity of computing founded repairs and consistent answers, showing that the complexity is not harder than computing “standard” repairs and answers.

Chapter 4 compares *active integrity constraints* and *revision programming*, another formalisms designed to describe integrity constraints on databases and to specify *preferred* ways to enforce them. We demonstrated that despite the differences in the syntax, and the lack of a simple correspondence between justified revisions and founded repairs, the two frameworks are closely related. The semantics for revision programs defines the concept of *justified revision*. A justified revision is a set of *revision literals*, an alternative way to model updates over a database, that can be inferred by means of the revision program and by the set of all atoms that do not change their state of *presence (in)* or *absence (out)* during the update process. We shown that each founded repair corresponds to a justified revision, but not vice-versa. We introduced two new semantics: one for active integrity constraints and one for revision programs. The first one allows us to compute a smaller set of repairs, the *justified repairs*, that correspond to justi-

fied revisions. The second one allow us to compute a wider set of revision, the founded revisions, that correspond to founded repairs. The introduction of these new semantics aligns the two formalisms showing that each of them is a notational variants of the other. We show that for each semantics the *shifting property* holds. Shifting consists of transforming an instance of a database repair problem to another syntactically isomorphic instance by changing active integrity constraints or revision programs to reflect the “shift” from the original database to the new one.

Chapter 5 has proposed a declarative semantics for view updating in the presence of existentially derived predicates and non flat integrity constraints, that prevents some of the anomalies previous approaches suffer from limiting the wide range of translations to those that are validated by the deductive database. More specifically, the novelty of the framework consists in the definition of a formal declarative semantics for view updating that allows to identify, among the set of all possible repairs, the *supported repairs*, that is the repairs whose actions are validated by the database or by other updates. In addition, the proposed repair semantics based on the rewriting of the deductive database and the update request into active integrity constraints, as specified in Chapter 3, considers, systematically and for the first time, the possible introduction of null value in the form they are present and treated in commercial DBMS. We have provided results on the soundness and completeness of the proposed approach and have also investigated the complexity of computing justified repairs showing that this is not harder than computing standard repairs. Two important issues, that could actually translate our research into practical applications, are left for further research. First, extensions may be introduced in order to select, in the presence of multiple justified repairs, the preferred ones, i.e. those that better satisfy some preference or quality criteria specified by the user. Moreover, further research is planned to investigate particular types of integrity constraints, implemented and maintained in commercial DBMS, such as primary keys and foreign key constraints for which the complexity of computing justified repairs is expected to reduce.

---

## References

1. ABITEBOUL, S., HULL, R., VIANU, V. (1995) Foundations of Databases. Addison-Wesley Publishing Co.
2. ABITEBOUL, S., DUSCHKA, O. M.(1998) Complexity of Answering Queries Using Materialized Views. Symposium on Principles of Database Systems, 254-263.
3. ABITEBOUL, S., VIANU, V.(1991) Datalog Extensions for Databases Queries and Updates. Journal of Computer and System Science, 43: 62-124.
4. ABITEBOUL, S., SIMON E., VIANU, V.(1990) Non-Deterministic Language to Express Deterministic Transformation. ACM Symposium on Principles of Database Systems, 215-229.
5. AFRATI, F., COSMADAKIS, S.S., YANNAKAKIS M.(1991) On Datalog vs. Polynomial Time. ACM PODS Conference, Delphi, Greece, 13-254.
6. AGARWAL, S., KELLER, A.M., WIEDERHOLD, G., SARASWAT, K.(1995) Flexible Relation: an approach for integrating data from multiple, possibly inconsistent databases. International Conference on Database Engineering, 495-504.
7. ALFERES, J. J., LEITE, J. A., PEREIRA, L. M., PRZYMUSINSKA, H., PRZYMUSINSKI, T. C.(2000) Dynamic updates of non-monotonic Knowledge Bases. Journal of Logic Programming 45, 1-3, 43-70.
8. ALFERES, J. J., PEREIRA, L. M., PRZYMUSINSKA, H., PRZYMUSINSKI, T. C.(2002) Lupsa language for updating logic programs. Journal of Logic Programming 138, 1-2, 87-116.
9. APT, K.R., BLAIR, H.A., WALKER, A.(1998) Towards a Theory of Declarative Knowledge. Foundations of Deductive Databases and Logic Programming, 89-148.
10. APT, K.R.(1990) Logic Programming. Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, 93-574.
11. APT, K.R., BOL, R.N.(1994) Logic Programming and Negation: A Survey. Journal of Logic programming, 19(20): 9-71.
12. ARENAS, M., BERTOSSI, L., CHOMICKI, J.(2000) Specifying and querying database repairs using logic programs with exceptions. International Conference on Flexible Query Answering, 27-41.
13. ARENAS, M., BERTOSSI, L. E., CHOMICKI, J.(1999) Consistent query answers in inconsistent databases. Symposium on Principles of Database Systems, ACM Press, 68-79.
14. ATZENI, P., TORLONE, R.(1992) Updating Intensional Predicates in Datalog. Data and Knowledge Engineering , 8:1-17.

15. BANCILHON, F., SPYRATOS, N.(1981) Update Semantics of Relational Views. *ACM Transaction on Database Systems*, 6(4): 557-575.
16. BARAL, C.(1997) Embedding revision programs in logic programming situation calculus. *Journal of Logic Programming* 30, 1, 83-97.
17. BARAL, C., KRAUS, S., MINKER, J.(1991) Combining Multiple Knowledge Bases. *IEEE-Trans. on Knowledge and Data Engineering*, 3(2):208-220.
18. BARAL, C., KRAUS, S., MINKER, J., SUBRAHMANIAN, V.S.(1991) Combining Knowledge Bases Consisting of First Order Theories. *International Symposium on Methodologies for Intelligent Systems*, 92-101.
19. BARAL, C., ZHANG, Y. (2001) On the semantics of knowledge update. *International Joint Conferences on Artificial Intelligence*, 97-102.
20. BASTA, S., FLESCA, S., GRECO, S., ZUMPANO, E. (1999) A System Prototype for the Evaluation of Queries on Materialized Views, 291-306.
21. BENTAYEB, F., LAURENT, D. (1998) View Updates Translations in Relational Databases. *Database and Expert Systems Applications*, 322-331, 1998.
22. BERTOSSI, L., PINTO, J. (2000) Specifying active rules for database maintenance. *Lecture Notes in Computer Science* 1773, 112-119.
23. BRAVO, L., BERTOSSI, L.(2006) Semantically Correct Query Answers in the Presence of Null Values. *Extending Database Technology Workshops*, 336-357.
24. BRASS, S., DIX, J.(1995) Disjunctive Semantics based upon Partial and Bottom-Up Evaluation *International Conference on Logic Programming*, 199-213.
25. BRASS, S., DIX, J., NIEMELÄ, I., PRZYMUSINSKI, T.C.(2001) On the Equivalence of the STATIC and disjunctive Well-founded Semantics and their Computation. *Theoretical Computer Science*, 258(1-2):523-553.
26. F. BRY(1997) Query Answering in Information System with Integrity Constraints. *IFIP WG 11.5 Working Conf. on Integrity and Control in Information System*.
27. CALÌ, A., CALVANESE, D., DE GIACOMO, G., LENZERINI, M.(2002) Data Integration under Integrity Constraints. *International Conference on Advanced Information Systems Engineering*, 262-279.
28. CALÌ, A., DE GIACOMO, G., LENZERINI, M.(2002) Models for information integration: Turning local-as-view into global-as-view. *International Workshop on Foundations of Models for Information Integration*.
29. CAROPRESE, L., GRECO, S., SIRANGELO, C., ZUMPANO, E.(2006) Declarative semantics of production rules for integrity maintenance. *International Conference on Logic Programming*, 26-40.
30. CAROPRESE, L., GRECO, S., SIRANGELO, C., ZUMPANO, E..(2006) Declarative Semantics of Production Rules for Integrity Maintenance. *Technical Report*. <http://www.info.deis.unical.it/~zumpano>.
31. CAROPRESE, L., ZUMPANO, E.(2006) A Framework for Merging, Repairing and Querying Inconsistent Databases. *Advances in Databases and Information Systems*, 383-398.
32. CERI, S., FRATERNALI, P., PARABOSCHI, S., TANCA, L.(1994) Automatic generation of production rules for integrity maintenance. *ACM Transaction on Database System* 19, 3, 367-422.
33. CERI, S., WIDOM, J.(1990) Deriving Production Rules for Constraint Maintenance. *International Conference on Very Large Data Bases*, D. McLeod, R. Sacks-Davis, and H. Schek, Eds. Brisbane, Australia, 566-577.
34. CHANDRA, A. K., MERLIN, P. M.(1977) Optimal implementation of conjunctive queries in relational data bases. *ACM Symposium on Theory of Computing*, 77-90.



35. CHAUDHURI, S., KRISHNAMURTHY, R., POTAMIANOS, S., SHIM, K.(1995) Optimizing queries with materialized views. *International Conference on Data Engineering*, 190-200.
36. CHEKURI, C., RAJARAMAN, A.(1997) Conjunctive Query Containment Revisited. *International Conference on Database Theory*, 56-70.
37. CHOMICKI, J.(2006) Consistent query answering: Opportunities and limitations. *International Conference on Database and Expert Systems Applications*, IEEE Computer Society, Washington, DC, USA, 527-531.
38. CHOMICKI, J., LOBO, J., NAQVI, S.(2003) Conflict resolution using logic programming. *IEEE Transactions on Knowledge and Data Engineering* 15, 1, 244-249.
39. CHOMICKI, J., MARCINKOWSKI, J., Minimal-change integrity maintenance using tuple deletions. *Information and Computation* 197, 1/2, 90-121, 2005.
40. CHOMICKI, J., MARCINKOWSKI, J., STAWORKO, S.(2004) Computing consistent query answers using conflict hypergraphs. *ACM International Conference on Information and Knowledge Management*, ACM Press, New York, NY, USA, 417-426.
41. CODD, E.F.(1970) A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377-387.
42. COLMERAUER, A., ROUSSEL, A.(1993) The Birth of Prolog. *IHOPL Preprints*, 37-52.
43. COSMADAKIS, C.C., PAPADIMITRIOU, C.H.(1983) Updates of Relational Views, *Symposium on Principles of Database Systems*, 317-331.
44. DANTSIN, E., EITER, T., GOTTLÖB, G., VORONKOV, A.(1997) Complexity and Expressive Power of Logic Programming. *IEEE Conference on Computational Complexity*, 82-101.
45. DAYAL, U., BERNSTEIN, P.A.(1998) On the Correct Translations of Update Operations on Relational Views. *ACM Transactions on Database Systems*, 13(4):486-524.
46. DUNG, P.M.(1996) Integrating Data from Possibly Inconsistent Databases. *International Conference on Cooperative Information Systems*, 58-65.
47. DUSCHKA, O. M., GENESERETH, M.R.(1997) Answering recursive queries using views”, *Symposium on Principles of Database Systems*, 109-116.
48. DUSCHKA, O. M.(1997) Query Planning and Optimization in Information Integration. PhD Thesis.
49. DUSCHKA, O. M., GENESERETH, M.R.(1997) Query planning in infomaster. *Symposium on Applied Computing*.
50. EITER, T., GOTTLÖB, G.(1992) On the complexity of propositional knowledge base revision, updates and counterfactuals. *Symposium on Principles of Database Systems*, 261-273.
51. EITER, T., GOTTLÖB, G.(1995) On the computational cost of disjunctive logic programming: propositional case. *Annals of Mathematics and Artificial Intelligence* 15, 289-323.
52. EITER, T., LEONE, N., SACCA, D.(1998) Expressive Power and Complexity of Partial Models for Disjunctive Deductive Databases. *Theoretical Computer Science*, 206(1-2): 181-218.
53. EITER, T., LEONE, N., MATEIS, C., PFEIFER, G., SCARCELLO, F. (1998) A Deductive System for Non-Monotonic Reasoning. *Logic Programming and Non-monotonic Reasoning*, 364-375.
54. EITER, T., GOTTLÖB, G., MANNILA, H.(1997) Disjunctive Datalog. *ACM Transactions on Database Systems* 22, 3, 364-418.
55. FAGIN, R., ULLMAN, J.D., VARDI, M.(1983) On the Semantics of Updates in Databases. *Symposium on Principles of Database Systems*, 352-365.

56. FARRÉ, C., TENIENTE, E., URPI, T.(2003) Handling Existential Derived Predicates in View Updating, International Conference on Logic Programming, 148-162.
57. FERNANDEZ, J.A., GRANT, J., MINKER, J.(1996) Model Theoretic Approach to View Updates in Deductive Databases. Journal of Automated Reasoning, 17(2): 171-197.
58. FITTING, M.(1985) A Kripke-Kleene Semantics for Logic Programs. Journal of Logic Programming, 2(4):295-312.
59. FITTING, M. (1994) On prudent bravery and other abstractions, Fitting, M. C., On prudent bravery and other abstractions.
60. FITTING, M.(1995) Annotated revision specification programs. Logic Programming and Non-monotonic Reasoning, 143-155.
61. FLESCA, S., GRECO, S.(2002) Answering queries using views. IEEE-Trans. on Knowledge and Data Engineering, 13(6):980-995.
62. FRATERNALI, P., PARABOSCHI, S.(1993) A Review of Repairing Techniques for Integrity Maintenance. Rules in Database Systems, 333-346.
63. FRIEDMAN, M., WELD, D.(1997) Efficient execution of information gathering plans”, Proc. Int. IJCAI.
64. VAN GELDER, A.(1998) Negation as Failure Using Tight Derivations for General Logic Programs. Foundations of Deductive Databases and Logic Programming, 149-176.
65. GELFOND, M., LIFSCHITZ, V.(1988) The stable model semantics for logic programming. International Conference on Logic Programming, R. A. Kowalski and K. Bowen, Eds. The MIT Press, Cambridge, Massachusetts, 1070-1080.
66. GELFOND, M., LIFSCHITZ, V.(1991) Classical negation in logic programs and disjunctive databases. New Generation Computing 9,365-385
67. GIANNOTTI, F., PEDRESCHI, D., SACCÀ, D., ZANIOLO, C.(1991) Nondeterminism in deductive databases. International Conference on Deductive and Object-Oriented Databases, 129-146.
68. GRAHNE, G., MENDELZON, A.O.(1999) Tableau Techniques for Querying Information Sources through Global Schemas. Lecture Notes in Computer Science, 1540, 332-347.
69. GRANT, J., SUBRAHMANIAN, V. S.(1995) Reasoning in inconsistent knowledge bases. IEEE Transactions on Knowledge and Data Engineering 7, 1, 177-189.
70. GRECO, G., GRECO, S., ZUMPARO, E.(2003) A logical framework for querying and repairing inconsistent databases. IEEE Transactions on Knowledge and Data Engineering 15, 6, 1389-1408.
71. GRECO, S., ZUMPARO, E.(2000) Querying inconsistent databases. International Conference on Logic for Programming and Automated Reasoning, 308-325.
72. GRECO, S., ZUMPARO, E.(2000) Computing Repairs for Inconsistent Databases. International Symposium on Cooperative Database System for Advanced Applications, 33-40.
73. GRECO, G., GRECO, S., ZUMPARO, E.(2001) A Logic Programming Approach to the Integration Repairing and Querying of Inconsistent Databases. Logic Programming, 17th International Conference, 348-364.
74. GRECO, G., SACCÀ, D.(1990) Negative Logic Programs. North American Conference on Logic Programming, 480-497.
75. GRECO, G., SACCÀ, D.(1995) Datalog Queries with Stratified Negation and Choice: from  $P$  to  $D^P$ . International Conference on Database Theory, 82-96.
76. GRECO, G., SACCÀ, D.(1999) Complexity and Expressive Power of Deterministic Semantics for Datalog<sup>-</sup>. Information and Computation, 153(1):81-98.
77. HEGNER, S.J.(1990) Foundations of canonical update support for close database views. International Conference on Database Theory, 422-436.
78. HEGNER, S.J.(2002) Uniqueness of Update Strategies for Database Views. Foundations of Information and Knowledge Systems, 230-249.

79. HERZIG, A., RIFI, O.(1999) Propositional belief base update and minimal change. *Artificial Intelligence* 115, 1, 107-138.
80. JOHNSON, D.S.(1990) A Catalog of Complexity Classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, A(2):67-161, Elsevier Science..
81. KATSUNO, H., MENDELZON, A. O.(1991) Propositional knowledge base revision and minimal change. *Artificial Intelligence* 52, 3, 263-294.
82. KANELLAKIS, P. C.(1991) *Elements of Relational Database Theory*. J. van Leewen. Vol. 2.
83. KIFER, M. LI, A.(1988) On the semantics of rule-based expert systems with uncertainty. *International Conference on Database Theory*, Springer-Verlag, 102-117.
84. KOLAITIS, P.(1990) The Expressive Power of Stratified Logic Programs. *Information and Computation*, Vol. 90, 50-66.
85. KOTIDIS, Y., SRIVASTAVA, D., VELEGRAKIS, Y.(2006) Updates Through Views: A New Hope. *International Conference on Data Engineering*, 2.
86. KOWALSKI, R. A.(1974) Predicate Logic as Programming Language. *IFIP Congress*, 569-574.
87. KOWALSKI, R. A.(1977) Logic for Data Description. *Logic and Data Bases*, 77-103.
88. KOWALSKI, R. A.(1979) Algorithm = Logic + Control. *Communications of the ACM (CACM)*, 22(7): 424-436.
89. KOWALSKI, R. A., SADRI, F.(1990) Logic programs with exception. *International Conference on Logic Programming*, 598-613.
90. KRISHNAMURTHY, R., NAQVI, S.(1988) Non-deterministic Choice in Datalog. *ACM Symposium on Principles of Database Systems*, 416-424.
91. KWOK, C.T. WELD, D.S.(1996) Planning to Gather Information. *AAAI National Conf. on Artificial Intelligence*, 32-39.
92. LEMBO, D., LENZERINI, M., ROSATI, R.,(2002) Source Incompleteness and inconsistency in information integration. *International Workshop on Knowledge Representation meets Databases*.
93. LENZERINI, M.(2002) Data Integration: A Theoretical Perspective. *Symposium on Principles of Database Systems*, 233-246.
94. LEVY, A.(1996) Obtaining Complete Answers from Incomplete Databases. *International Conference on Very Large Data Bases*, 402-412.
95. LEVY, A.Y., MENDELZON, A.O., SAGIV, Y., SRIVASTAVA, D.(1995) Answering Queries Using Views. *Symposium on Principles of Database Systems*, 95-104.
96. LEVY, A.Y., RAJARAMAN, A., ULLMAN, J.D.(1996) Answering queries using limited external query processors. *Symposium on Principles of Database Systems*, 227-237.
97. LEVY, A.Y., RAJARAMAN, A., ORDILLE, J.J.(1996) Querying Heterogeneous Information Sources Using Source Descriptions. *International Conference on Very Large Databases*, 251-262.
98. LIBERATORE, P.(2000) The complexity of belief update. *Artificial Intelligence* 119, 1-2, 141-190.
99. LIFSCHITZ, V.(1988) On the Declarative Semantics of Logic programs with negation. *Foundations of Deductive and Logic Programming*, Morgan Kaufmann, Los Altos, CA, J. Minker, 177-192.
100. LIFSCHITZ, V., WOO, T.(1992) Answer sets in general nonmonotonic reasoning. *International conference on principles of knowledge representation and reasoning, KR '92*, San Mateo, CA, Morgan Kaufmann, 603-614.
101. LIN, J.(1996) A semantics for reasoning consistently in the presence of inconsistency. *Artificial Intelligence* 86, 1, 75-95.

102. LIN, J.(1996) Integration of Weighted Knowledge Bases. *Artificial Intelligence*,83(2):363-378.
103. LIN, J., MENDELZON,A. O.(1998) Merging Databases Under Constraints. *International Journal of Cooperative Information Systems*, 7(1):55-76.
104. LIN, J., MENDELZON,A. O.(1999) Knowledge Base Merging by Majority. *Dynamic Worlds: from the frame problem to knowledge management*, Kluwer.
105. LLOYD, J.W.(1987) *Foundations on Logic Programming*. 2<sup>nd</sup> Edition, Springer.
106. LLOYD, J.W., TOPOR, R. W.(1985) A Basis for Deductive Database Systems. *Journal of Logic Programming*, 2(2): 93-109.
107. MAREK, V. W., TRUSZCZYŃSKI, M.(1991) Autoepistemic logic. *Journal of the ACM*, **38**, 588-619.
108. MAREK, V. W.,TRUSZCZYŃSKI, M.(1998) Revision programming. *Theoretical Computer Science* **190**, 241-277.
109. MAREK, V. W., PIVKINA, I.,TRUSZCZYŃSKI, M.(1998) Revision programming = logic programming + integrity constraints. *Computer Science Logic*, 73-89.
110. MAREK, V. W., PIVKINA, I.,TRUSZCZYŃSKI, M.(2002) Annotated revision programs. *Artificial Intelligence* 138, 1-2, 149-180.
111. MAREK, V. W.,TRUSZCZYŃSKI, M.(1998) Revision programming. *Theoretical Computer Science* 190, 2, 241-277.
112. MAYOL, E., TENIENTE, E.(1999) A Survey of Current Methods for Integrity Constraints Maintenance and View Updating. *International Conference on Conceptual Modelling / the Entity Relationship Approach Workshop*, 62-73.
113. MAYOL, E., TENIENTE, E.(2003) A Review of Integrity Constraints Maintenance and View Updating Techniques. *Data and Knowledge Engineering*, 61-103.
114. MEDEIROS, C., ANDRADE, M.(1994) Implementing integrity control in active databases, *Journal of Systems and Software*, 27(3):171-181.
115. MINKER, J.(1982) On Indefinite Data Bases and the Closed World Assumption. *Conference on Automated Deduction*, 292-308, 1982.
116. MINKER, J.(1988) Perspectives in Deductive Databases. *Journal of Logic Programming*, 5(1):33-60.
117. PAPADIMITRIOU, C. H.(1994) *Computational Complexity*. Addison-Wesley.
118. PIVKINA, I.(2001) *Revision programming: a knowledge representation formalism*. PhD thesis, Department of Computer Science, University of Kentucky.
119. PRZYMUSINSKI, T.C.(1990) Well-founded Semantics Coincides with Three-valued Stable Semantics. *Fundamenta Informaticae*, 13:445-463.
120. PRZYMUSINSKI, T.C.(1991) Stable Models and Non-Determinism in Logic Programs with Negation. *Special issue of the New Generation Computing Journal*, 9(3):401-424.
121. PRZYMUSINSKI, T.C.(1995) Static Semantics for Normal and Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12(2-4):323-357.
122. QIAN, X.(1996) Query Folding. *International Conference on Data Engineering*, 48-55.
123. RAJARAMAN, A., SAGIV, Y., ULLMAN, J.D.(1995) Answering queries using templates with binding patterns. *ACM Symposium on Principles of Database Systems*, 105-112.
124. RAMAKRISHNAN, R., SAGIV, Y., ULLMAN, J. D.,VARDI, M. Y.(1989) Proof-Tree Transformation Theorems and Their Applications. *Symposium on Principles of Database Systems*, ACM Press, 172-181.
125. REITER, R.(1982) Towards a Logical Reconstruction of Relational Database Theory. *On Conceptual Modelling (Intervale)*, 191-233.
126. REITER, R.(1978) On Closed World Data Bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, 55-76.

127. SACCÀ, D.(1997) The Expressive Powers of Stable Models for Bound and Unbound Datalog Queries. *Journal of Computer and System Science*, 54(3):441-464.
128. SACCÀ, D., ZANIOLO, C.(1990) Stable models and non-determinism in logic programs with negation. *ACM Symposium on Principles of Database Systems*, 205-217.
129. SACCÀ, D., ZANIOLO, C.(1997) Deterministic and Non-Deterministic Stable Models. *Journal of Logic and Computation*, 7(5):555-579.
130. SARAIYA, Y.(1991) Subtree elimination algorithms in deductive databases. PhD Thesis, Stanford University.
131. SHMUELI, O.(1987) Decidability and expressiveness aspects of logic queries. *ACM Symposium on Principles of Database Systems*, 237-249.
132. SHMUELI, O.(1993) Equivalence of Datalog Queries is Undecidable. *Journal of Logic Programming*, 15(3):231-241.
133. SHU, H.(2000) Using Constraint Satisfaction for View Update. *Journal of Intelligent Information Systems*, 15(2): 147-173.
134. SUBRAHMANIAN, V. S.(1994) Amalgamating knowledge bases. *ACM Transaction on Database Systems* 19, 2, 291-331.
135. TARSKI, A.(1955) A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, Vol.5, 285-309.
136. TENIENTE, E., OLIVÉ, S.(1995) Updating knowledge bases while maintaining their consistency. *VLDB Journal*, 4(2):193-421.
137. TODD, S.(1977) Automatic constraint Maintenance and updating defined relations. *IFIP Congress*, 145-148.
138. TORLONE, R., ATZENI, P.(1991) Updating Deductive Databases with Functional Dependencies. *International Conference on Deductive and Object-Oriented Databases*, 278-291.
139. ULLMAN, J. D.(1988) *Principles of Database and Knowledge-Base Systems*. Vol. 1-2. Computer Science Press, Potomac, Maryland.
140. ULLMAN, J. D.(1997) Information Integration Using Logical Views. *International Conference on Database Theory*, 19-40.
141. ULLMAN, J.D.(2000) Information integration using logical views. *Theoretical Computer Science*, 239(2): 189-210.
142. VAN EMDEN, M.H. KOWALSKI, R. A.(1976) The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733-742.
143. VAN GELDER, A., ROSS, K., SCHLIPF, J. S.(1991) The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620-650.
144. VARDI, M.Y.(1982) The Complexity of relational query languages. *ACM Symposium on Theory of Computing*, 137-146.
145. WIJSEN, J.(2003) Condensed representation of database repair for consistent query. *International Conference on Database Theory*, 378-393.
146. WINSLETT, M.(1990) *Updating logical databases*. Cambridge University Press, New York, NY, USA.