On Updates of Epistemic States

Belief Change under Incomplete Information

Doctoral Thesis
(Dissertation)

to be awarded the degree of
Doctor rerum naturalium (Dr.rer.nat.)

submitted by

M.Sc. Juan Carlos Acosta Guadarrama
from Toluca, Mexican United States

approved by

the Faculty of Mathematics/Computer Science and Mechanical Engineering,
Clausthal University of Technology,

Date of oral examination
18.12.2009

Chairperson of the Board of Examiners:	Prof. Dr. G. Zachmann
Chief Reviewer:	Prof. Dr. J. Dix
Reviewers:	PD. Dr. habil. W. Jamroga Prof. Dr. S. Hartmann

Abstract

In this dissertation I present some aspects of belief-change theory and representation of knowledge as one of the main theoretical basis to formulate semantics for updates of logic programs. Firstly, there is an introduction to relevant principles and postulates, like the classical belief-revision formulation and a following proposal to make a difference between belief revision and updates. Next, there is a survey of some few proposals to update logic programs that are the main motivation for this thesis. Finally, I present a progressive approach that overrides the problems pointed out in other alternatives, and that meets most of the principles here introduced.

In particular, revising and updating knowledge bases is an important problem in knowledge representation and reasoning. It has led to various proposals for updating logic programs, specifically with respect to the well known answer-sets semantics. However, most of these approaches are based on the causal rejection principle, which leads to counter-intuitive behaviour. The proposed approach in this thesis is a semantics for abduction known as generalised answer sets, which allows one to choose potential models, without changing the semantics of the original given update programs. With generalised answer sets one can actually formulate semantics for updates that consist in choosing between generalised models that satisfy an intended set of properties and overcome certain problems from other approaches. Weak Irrelevance of Syntax and Strong Consistency are two of the main properties an update semantics should manifest, which are a keystone to overcome the mentioned problems.

Finally, as an important component of logic programming and as a useful tool in the classroom, this work also provides the research community with online solver prototypes that help close the gap between theory and practice. These automatic testbeds make the semantics more accessible, and open up a path with a solid component for further more-complex prototypes of knowledge management.

COPYRIGHT

Acknowledgements

I would like to acknowledge the National Council of Science and Technology of Mexico for a doctoral grant, which has funded a large part of this project.

Software Tools

The typesetting of this document has been accomplished with help of LaTeX2e, BibTeX, and ACM's bibliography style, all running on TeXShop on Mac OS™ systems. Most of the implementations are running on Apache Mac OS™or Linux servers.

Reviewers

This work has also been accompanied by close reviews from PD. Dr. W. Jamroga, as well as some other anonymous reviewers from workshops and conferences. Prof. M. Osorio also played a key role in publishing one of the papers.

Juan Carlos Acosta Guadarrama

TU-Clausthal, 2009

Contents

List of Tables

Preface

The original title of my project was *A Language for Beliefs and Knowledge Representation*. However, such a language requires many subproblems to be solved, that range from theoretical and philosophical view points up to practical and industrial applications. One of the most interesting problems is an intermediate solution based upon a strong theoretical foundation that might model a large number of situations in order to maintain knowledge bases in an automatic way.

This dissertation consists of a study of identified problems in several existing approaches and comprises two solutions generalised into a set of properties. There are two main streams to explore in this study. On the one hand, this work also consists of a small collection of the most relevant (to the author's opinion) and well-known principles for knowledge evolution. On the other hand, it recaps logic programming by means of Answer Sets semantics as an alternative and practical framework to solve problems of knowledge representation. Nevertheless, the determination to remain in both streams is by no means an exhaustive decision, but focused on research interests and existing background. As a result, the author's efforts have been stressed on producing a sound and robust framework that overcomes the presented drawbacks of other particular proposals.

To begin with, the author of this dissertation gives a general introduction in Chapter 1 including basic concepts of beliefs, knowledge representation and reasoning, related fields of research around it and a general picture of existing problems. The chapter includes some background on the role of logic programming to represent knowledge and some of its limitations. Next, there is a background of current proposals to representing dynamic knowledge and a statement of the broad problem to be addressed in this research work, together with the pursued general goal and a proposal. Finally, there is a list of contributions followed by a brief description of the publications that constitute the final dissertation.

Chapter 2 is an overview of the main *theoretical foundation* of this thesis: *logics*, *postulates* and non-monotonic reasoning, which shall be recapitulated in subsequent chapters. The chapter is a collection of particular principles that should be observed when representing knowledge evolution. They include *non-classical logics* that have a strong relation to logic programming, postulates for *belief revision* and for *updates*, as well as relevant particular *properties*.

The next foundation layer for the proposals in this dissertation is a general framework in Chapter 3, which includes logic programming, notation, and some theoretical

results from the literature, useful in upcoming chapters. The suggested language to pursue the representation of dynamic knowledge is the language of a well-known semantics, named *Answer Set Programming* (ASP) or *Stable Models Semantics*. Finally, the chapter comprises a particular instance of logic programming called *abduction* modelled by *generalised answer sets* and two semantics of *preferences*.

Chapter 4 is the initial *motivation* of this work and describes (to my knowledge) the most relevant and well known approaches to this thesis and points out some of their features and limitations to represent evolving knowledge. The selection of such proposals is based upon relevance and research interests, like Answer Set Programming and their impact, and the survey is by no means exhaustive.

Chapter 5 is a summary of the initial motivation for this thesis, where problems found in other approaches, up to then, are put together into a list of counterintuitive examples that I call *observations*. The function of such observations is to highlight key differences with the approaches in Chapter 4 and the proposals of this dissertation.

The main contribution of this thesis is distributed along Chapters 6–8. Such a contribution starts with the basic concept of a mechanism of *relaxation* to weaken beliefs (Chapter 6) and its relation with *preferences*, followed by a proposal to perform *updates sequences* of logic programs. Finally, Chapter 8 comprises a generalised approach to perform *iterated updates* along with further properties and variants for specific needs. In other words, this research work provides a combination of strong theoretical foundation on well-known principles and fundamental properties shared mainly in Chapters 6 to 8, that overcome problems of redundant information and provide a framework with *intuitive behaviour* to represent evolving knowledge. Nevertheless, the goal of this research does not mean an exhaustive comparison with alternative existing approaches, which would be matter of further research and out of the scope of this current proposal. The reader may find a list of properties and principles introduced along this dissertation summarised in a table in Appendix A. That summary is a comparison between the approaches introduced in Chapters 6–8 and also includes a brief glimpse to the other alternatives introduced in Chapter 4. In addition, each chapter includes a description of a prototype that implements the semantics and the employed tools to compute it, and they are summarised and collected in Appendix B, which is a list of solvers and prototypes with their respective links of implementations.

Last, Chapter 9 is a final general view of the addressed problem, solution, contributions, potential applications, the role of a theoretical foundation and solvers, and general discussions.

Chapter 1

Introduction

Revising and updating knowledge is an important problem in knowledge representation and reasoning that has led to various proposals for updating logic programs, in particular with respect to the well-known answer-sets semantics. However, most of these approaches have been based on a causal rejection principle, which sometimes leads to counter-intuitive behaviour. The approach proposed in this thesis is a semantics for abduction known as minimal generalised answer sets that allows to choose candidate models without changing the semantics of the original given update programs. Such semantics are formulations for updates that consist in choosing between generalised models that satisfy several structural properties and thus overcomes certain problems of earlier approaches. Some of the main properties an update process should exhibit are *independence from syntax* and *logical-model reliance*.

Finally, as an important component of logic programming, this work also provides the research community with online solver prototypes that help close the gap between theory and practice. These automatic testbeds make the semantics more accessible and opens up a path with a solid component for more-complex prototypes of knowledge management.

The present chapter consists of a short background from which the topic of this dissertation emerges, and is organised as follows. Firstly, it includes a recap to its main stream of the earliest years of *Artificial Intelligence* and *Logic Programming*. Secondly, it includes a description how problems from knowledge representation and reasoning came about, and how they were to be solved by these new disciplines. Next, it comprises a basic background of *belief change* and how logic programming semantics have taken up the problem. Then, the chapter includes a particular example that motivated this research. Finally, it comprised the goals and contributions of this work, with a list of publications on which this thesis is built.

1.1 Knowledge and Belief Representation

To begin with, *Knowledge* is defined in the literature as a *justified true belief* an entity may have. On the other hand, a *belief* is understood in the literature as a *propositional attitude* represented in a form of sentence or statement called *proposition*. These broad definitions are of particular interests to philosophers and logicians and have gain influence in computer science over the last few decades, due to the huge advance in power and storage of computer systems, as well as its potential to manage symbolic operations.

Knowledge representation is by no means trivial tasks, and the particular theory of information to deal with it is called *Epistemology*. This discipline is one of several theories of information, which is known in the literature as the *theory of knowledge* and has to do with *what* propositions are true and what to *reason* about them, obviously underpinned by a *logic system*. In contrast, *procedural knowledge*, which has also the popular term: *know-how*, deals with a set of steps or maybe a *fuzzy process* on how to perform a specific operation, typically supported by boolean or continuos bases. There are other notions of knowledge in the literature that shall be out of scope of this thesis, including imperative and procedural ones, to mention a few.

Some particularly specialised fields in computer science for which this notion of knowledge is relevant, are *Knowledge Representation and Reasoning*, *Common-sense Reasoning* and *Nonmonotonic Reasoning*. Those specialised fields come out of more general and earlier *theories of information* like *Artificial Intelligence* and *Logic Programming*, whose *Holy Grail* has been the ambitious achievement of an intelligent *autonomous* computer than may take decisions with little or null human intervention. The merge of these two produced yet a more specialised applied discipline called *Expert Systems*, which may be considered an ancestor of modern agent systems.

Commonsense Reasoning and Logic Programming, in particular, sprang up as two major fields in computer science to represent knowledge. The role of *mathematical logic* in these disciplines is particularly important for providing unambiguous succinct formal languages to effectively manipulate propositions.

Finally here comes the concept of an *agent* that should manipulate such knowledge. An agent may be described as an intelligent entity capable of being an assistant to other agent(s) in decision making, because of its potential precision to operate, and vast knowledge on a particular task or topic. Later, several architectures arose and made possible to classify agents and multi-agent systems into several categories, where *logical, belief-desire-intention, and intelligent agents* are of particular interest in knowledge representation and reasoning.

1.2 A Question of Principles

The purpose of this section is to introduce a general background of basic concepts in representing and changing knowledge. It starts with a little history of the problem and concludes with particular proposals to found this thesis.

1.2.1 Logic in Artificial Intelligence

Knowledge representation and reasoning is a very old activity that dates back to *Aristotle*, who started formulating principles to achieve a correct reasoning from natural language. His activity consisted in establishing relations between an agent and a proposition by means of a *symbolic representation*. By following a set of principles, the agent could manipulate symbols and draw conclusions. Such principles later derived in a set of axioms that became a logic system better known as *Aristotelian logic* or *classical logic*, which is well-known and widely used in current technology.

After many centuries of its conception, classical logic started to show many limitations so as to be useful in the solution of *any* problem. Some of such limitations is to represent time and other resources, for which several other logic systems were formulated. One of those systems is *intuitionistic logic*, due to *Brouwer*, relevant to this thesis for its computing meaning.

In 1907 the Dutch mathematician L.E.J. Brouwer questioned the principles of Classical Logic, that were considered untouchable until then [Brouwer, 1907]. In particular, the principle of the excluded middle ($\alpha \vee \neg\alpha$) was criticised because it enforces that a truth-value is assigned without even knowing whether a constructive proof for α (or its negation) can be obtained. In other words, intuitionism takes only the "safe" principles from Classical Logic and gets rid of those for arbitrary undefined witnesses [Mints, 2000].

There are other relevant logic systems derived from intuitionistic logic, like Gödel's $\mathcal{G}_3$, Heyting's *Here-and-There* $\mathcal{H}_{HT}$, Nelson's logic $\mathcal{N}$ and $\mathcal{N}_2$ that specify notions of proof, truth values and negations, also useful in knowledge representation that constitute part of a solid theoretical basis for this thesis.

Besides these logic systems, there are alternative ones designed to specify dynamic theories, like non-monotonic logics and *Reiter*'s *Default Logic* that, although very relevant to *logic programming*, shall not be introduced in this thesis. Some others more related to knowledge and beliefs and less to logic programming are *modal logic*, *epistemic logic* that shall not be introduced in this work either.

1.2.2 Knowledge Incompleteness

One general and very old principle is the one that states knowledge as boundless, and typical logic systems assume a *closed world* and *complete knowledge* of a given problem. Of course, there are many current applications of such complet knowledge and classical logic like digital circuits or game theory. However, there are many more problems of *incomplete knowledge* in an *open world* that require systems to model them. *Nonmonotonic logics* and, particularly, *nonmonotonic reasoning* are specialised fields of research that study such problems.

Nonmonotonic reasoning emerges when problems from commonsense reasoning considered unexpected possibilities, which may contradict current conclusions. Such problems come especially from *planning* domains, where propositional logic isn't expressive enough.

On the one hand, classical logics and mathematics are essentially *monotonic*, which means that their foundational theories evolve by building up more principles that do not contradict previous ones. That is to say, current principles are always implied in expanded versions.

On the other hand, nonmonotonic theories evolve by following a classical *inertia principle*, stating that conclusions remain unless new information contradicts them. Accordingly, there are new concepts introduced that can be modelled by such formalism, like *normality*, *abnormality*, *defaults*, *expectations* and *inertia*, as well as *incompleteness*, *closed-world assumption* and *uncertainty*. As a result, not only can a nonmonotonic theory expand, but also contract.

In summary, incomplete knowledge is a basic principle assumed in this research work. There are other related approaches like *probabilistic reasoning*, *paraconsistent logics* and *Default Logic* that lie beyond the scope of this thesis.

1.2.3 Theory Change

Nonmonotonic reasoning then implies a dynamics in formulation of theories, although it does not imply the procedure to change them. Inevitably a new mechanism to perform such changes is needed. *belief change* is a new field of research that studies such changes under some basic principles of *truth maintenance* and *minimal change*.

One of the most studied formalisms is called *belief revision*, dated back to the 1980's as a new framework to represent changes in theories, which was something missing in nonmonotonic reasoning. There is a typical theory of change from Alchourrón et al. commonly known as the AGM-postulates.

There are three particular basic problems to solve in belief change. One is how to represent beliefs, already sketched in Section 1.1. Secondly, another problem is how to incorporate new observations to background knowledge. Finally, the problem of a changing environment is also to be considered.

One of the earliest ways to represent beliefs is by means of infinite *belief sets* of sentences logically closed in a formal language, on which AGM theory was defined. As everything, this conception has many advantages and disadvantages. Some of its advantages are that it is general-enough to idealise any belief-change system. However, the ultimate goal of artificial intelligence assumes an automatic agent with a *finite knowledge base*.

After AGM-postulates, there were further paraphrases to make them suitable to particular problems. A particular representation of beliefs is due to Katsuno and Mendelzon (*KM postulates* hereafter), who proposed a finite way to represent beliefs called (computer-based) *belief bases* —from now on *knowledge bases*— with the necessary paraphrase to the original postulates. There is yet another relevant paraphrase of the postulates by Darwiche and Pearl that suggests representing beliefs by means of *epistemic states* rather than belief bases.

Besides considering finite sets of beliefs, an epistemic state also includes the particular strategy to perform belief revision, which is the representation adopted in this

thesis.

Regarding the second problem of belief change, some of the most accepted and original strategies to perform revision on an agent's *belief set* is a set of *postulates* that specify changes to a belief set, better known as *AGM-postulates*. Such specifications are strategies to incorporate new information coming up form the environment, by means of three basic operations: *expansion*, *contraction* and *revision*. The problem is crucial in this thesis, and is fully introduced in Section 2.3.

The basic operations performed by AGM-postulates, as mentioned above, are expansion, contraction and revision. *Expansion* consists in adding a new sentence to the knowledge set without any concern about its conflict with other sentences. In contrast, in an operation of*contraction* a specific sentence is removed from the knowledge set so that it is no longer implied by the resulting set of sentences, also known as *derogation*. Finally, *revision* is an expansion of a new sentence and contraction of its possible inconsistent sentences from the knowledge set, which is also called *amendment*. Such a process of contraction and expansion has a formal definition and is known in the literature as *Levi's identity*.

The third problem for belief change to deal with is the *environment*. The environment may be defined as the state of the universe in which a specific agent is acting. Traditionally, an agent makes *belief revision* when incorporating new information from a *static environment*. On the other hand, it performs *belief update* when incorporating from a *changing environment*. Katsuno and Mendelzon formalised the difference between the two of them by means of a new set of postulates known as *belief update postulates*, and are fully introduced in Section 2.4.

Both of the two types of environment are useful for specific applications, and an update setting seems to be more general and what *real world* is. In such a case, an agent would perform updates of "perfect" (coherent) information from scratch without need to make any revision at all. And that would require a *perfect agent* as well, whose observations are never mistaken. However, it might be the case that real world never changes and that agents just cannot know everything!

In practice, *electronic circuits* have to be constrained to a small-scale static environment. In such a case, an evolving agent formulating a theory out of a circuit would make revision when a contradictory observation occurred. Still the observation would come from the closed world of a circuit and would mean that the theory has a *bug*. Then, the agent would act accordingly by retracting the necessary information to remove the bug so that the theory is correct. But what if the circuit is *extended*[1] with new features to innovate in the world of technology? Of course, such events normally do not happen so often in industry so that belief update might be interesting to them now, but the problem still exists. What is more, electronic circuits and even the digital ones are known to consider *uncertainty principles* to operate!

In accordance with the theses of knowledge incompleteness, uncertainty and of commonsense reasoning, a correct combination of principles from both belief revision and belief update must be the most optimal framework to be able to solve real-wold prob-

[1]Here "extended" may mean either update or revision!

lems.

1.3 Representing Knowledge with Logic Programs

A famous concept of *Logic Programming* was first introduced by Robert Kowalski in the 1970s with the aim at having *automated theorem proving* and a problem-solving system. Such a system should be founded on logic and produced its first results with Colmerauer and `Prolog`. Another contemporary system with similar goals and features is `LISP`, which shall be out of the scope of this thesis.

Both `Prolog` and logic programming differ from traditional *imperative programming* in the specifications of a problem to be solved, rather than the necessary steps to find a solution. As a result, the task to work it out is left up to a particular underlying theorem prover. This singular way of specifying a problem to solve is known as *declarative programming*, and gave fundament to the classical equation from Kowalski:

$$algorithm = logic + control$$

Accordingly, `Prolog` introduced new concepts like *negation-by-failure*, rules, facts, goals, assertions, retractions, and with them new problems. To begin with, such negation-by-failure did not correspond to the classical negation in logic. In fact it came up from a particular way of resolution at the lower control level, which also obliged the programmer to be aware of a particular order of rules. Consequently, choosing a wrong order might lead to an endless loop, and other new control statements at the meta-language were in order. So, the new programming paradigm was not that declarative after all.

Then logic programming led to other fields of research called *satisfiability of logic formulas*, *theorem-proving* itself, *auto-epistemic logic*, and again *Default Logic*.

Stable Models (or SM in short) was a result of research on auto-epistemic logic and default logic by Gelfond and Lifschitz. One of the main goals in the research was to find a foundation of *provability* to `Prolog`'s *deduction* mechanism [Gelfond and Lifschitz, 1988; Pearce, 1999a]. Not much later, Lifschitz and Woo formulated an extension to Stable Models called *Answer Set Programming* (or ASP in short) that has become a little more expressive by allowing *two kinds of negation*: a founded study of `Prolog`'s negation-as-failure was then *default negation*, and *strong negation* would have an equivalent of *classical negation*. Although strong negation is easy to be "captured" in Stable Models, ASP programs with such negation are more succinct, and their models are then composed by literals rather than only atoms, like in Stable Models.

Many other ways to define both SM and ASP may be found in the literature; namely, default theory, non-monotonic reasoning, logical, planning, algorithmic, declarative, and others. Each of them is a characterisation of ASP or SM with a particular framework of interest. The particular approaches proposed throughout this thesis are logical and non-monotonic, as major building blocks of a *solid theoretical basis* to correctly represent knowledge.

Finally, there are at least two major and efficient competitive *proving solvers* for ASP, SMODELS [Niemela and Simons, 1997] and DLV [Calimeri et al., 2002; Leone et al., 2006], which contribute to the equation of logic programming proposed by Kowalski, and significantly help in the research of new properties and rapid prototyping of complex applications.

There are yet other comparable proposals to do logic programming with advantages and disadvantages, like *Well-founded Semantics* or simply WFS, by Van Gelder et al.. Although it has been deeply studied in the literature [Dix, 1995a,b] it lies beyond the focus of this thesis.

1.4 Updating Theories with Logic Programming

Logic programming then has become a widely supported field to represent knowledge, and one major challenge both in Prolog and ASP is how to change a logic program. Some preliminary and fundamental studies in ASP aimed at changes in logic programs derived several concepts of equivalence, namely, *weak* and *strong equivalence*, due to Lifschitz et al.. These concepts are very important to be considered when changing non-monotonic theories and shall be recapitulated in the next chapters.

Representation of knowledge by means of logic programs has been traditionally static, even after Prolog introduced statements in the meta-language to self-change its propositions, namely, assert and retract. The problem with the propositions, however, is that they were too general so as to be useful: their simple operations themselves are executed without any condition, producing *unpredicted effects* in a knowledge base, also known as *side effects*. Although there are quite many applications of *static knowledge*, like circuit problems mentioned in Section 1.2.3, both languages do not seem to be enough to correctly represent *dynamic knowledge* by themselves, in a *dynamic environment*. So, one still has to work on an extension to either.

1.4.1 Semantics for Updates

In order to represent dynamic knowledge, one of the first problems to work out is how to avoid inconsistencies due to potentially contradictory new information from the environment [Alchourrón et al., 1985; Katsuno and Mendelzon, 1991b; Zhang, 1995]. As a result, various researchers proposed a particular semantics to specify changes in knowledge bases, satisfying certain properties and postulates [Alferes et al., 2005; Eiter et al., 2001, 2002, 2005; Zhang, 2006]. What is more, some few of them were beyond and made complete frameworks of existing different approaches to compute them, like [Eiter et al., 2005; Zhang, 2006].

From the various proposals to update programs, Zhang classified them in three types: syntax-based, model-based and combined semantics. According to Zhang, a *model-based semantics* is characterised by the particular semantics of the logic programs under consideration. In contrast, a *syntax-based semantics* is characterised by the resulting logic

programs from the update. Finally, a *combined semantics* integrates both methodologies. Like everything, each of them has advantages and disadvantages, as shown later in Chapter 4 that shows how to update under different criteria.

Zhang's approach combines advantages from the two methodologies and propose three general principles to meet: *contradiction elimination*, *conflict resolution* and *syntactic representation*. The first principle is one of the most obvious in semantics for updates and belief revision, which should be real by preserving a *minimal-change principle* and a proper *justification*. On the other hand, *conflict resolution* has to do with potential future *contradictions* an update might yield because of the introduction of the two kinds of negations in logic programs —strong and default negation. Finally, once the process meets the two main goals, the author argues that a proper semantics should also *preserve* as many as possible of the original rules from the *updating knowledge base*.

Unfortunately, their approach seems too specific and thus appropriate to solve particular sorts of problems. As a result, there is a lack of general properties in their approach, as well as counterintuitive behaviour in certain circumstances. Such is the case of most of the rest of the semantics under consideration, even though Eiter et al. introduce a vast amount of general properties from the literature.

In consequence, this thesis claim is that combining both a solid theoretical basis and a proper operational semantics, one may overcome counterintuitive behaviour to correctly represent dynamic knowledge. The combination proposed is a generalisation of declarative logic programming to represent dynamic knowledge, consisting of a strong background of principles, and a semantics based upon both models and syntax. Because well-known principles are supported by decades of studies, debate and coherent languages, one may expect an intuitive behaviour from a framework that satisfies them. On the other hand, by having a combined approach of models and syntax in an operational semantics, the result must be general-enough for a wider range of applications, then problems should be represented in a more succinct, natural and efficient way.

Considering a solid background of principles when performing updates of logic programs is not new. In fact, Eiter et al. is one of the first or even the first to realise a deep study from the literature of belief change of logic programs, in particular in ASP, most of which will be introduced in Chapter 2. Other recent studies have been on belief-revision postulates and logic, by Osorio and Cuevas, characterised with operational semantics, but still specific, limited and inconclusive from the operational point of view.

Once a solid theoretical background is defined, one should be able to define a framework for updates of logic programs with intuitive behaviour, robust enough to keep a knowledge base against unexpected information that might jeopardise its integrity. In addition, formulating a semantics with such basis should have a number of *advantages*. First, the semantics may be more unambiguous for its logic foundation. Because it is unambiguous, the semantics should satisfy its own constraints and specifications, leading to a robust semantics. A robust system is reliable against unforeseen situations that make it more autonomous. Finally, after combining two lines of research from a taxonomical study, the semantics should be easier to integrate to more complex systems and thus should be general-enough to solve a wider range of problems.

Part of such theoretical foundation has already been established by some authors like Eiter et al. and Osorio and Cuevas, who employ ASP as a common framework. The former, as mentioned before, introduce a large set of principles, in particular from *belief change* and *non-monotonic* literature. On the other hand, Osorio and Cuevas have introduced a *logical framework* to characterise updates.

On the operational side, there are also background proposals with implementations called *solvers*. For example, of the earliest approaches to update logic programs is due to Alferes et al., who propose three basic principles to meet: *inertia*, *persistency* and *causal rejection*. The semantics that satisfy them is called *Dynamic Logic Programming* (or simply DyLP) and derived interesting *logic-programming languages* like LUPS and EVOLP. Most of them have been implemented both in ASP and WFS, and have also been inspiration to other update approaches like Eiter et al.'s who proposes an alternative semantics based upon DyLP's causal rejection.

Unfortunately, such principle of *causal rejection of rules* is particularly highly dependent on syntax and yields to counterintuitive behaviour under certain general circumstances, introduced in Section 4.2 and studied in Section 6.1.

There is an alternative approach proposed by Sakama and Inoue, which is general-enough to overcome some of the counterintuitive behaviour from such a syntax dependency. They even establish three basic operations in their semantics: *Consistency Restoration*, variant and *invariant knowledge* updates. A main objection against this kind of *syntax-based semantics* is the lack of a semantic foundation to justify its updates [Zhang, 2006], and that might conflict with the semantics that performs the update operation. Instead, Sakama and Inoue justify their updates with a particular *extended abductive framework*, which is still a specific problem and then leads to an absence of update characterisation.

Despite the existence of many principles and approaches to update logic programs, and existing solvers for most of them —summarised in Appendix B— some counterintuitive results still persist as a common problem, and is a case study in the survey of Chapter 4. As a result, each of the introduced approaches is appropriate for particular contexts and the need of the "right" semantics to represent *dynamic knowledge* still persists. A major contribution of the research work in this thesis is a careful analysis of typical semantics for updates in Chapter 4, as well as a summary of counterintuitive examples to these approach, presented in Chapter 5, which may be employed as a sort of "*benchmark*" for further studies on update semantics and to support claims in this thesis. A correct combination of advantages from the background literature, as well as a complete basis of belief-change theory should yield to a *general semantics* that may overcome at least the mentioned counterintuitive behaviour, proposed in Chapter 8.

1.4.2 Problem Definition

The following example has been the initial motivation to carry out the research work of this thesis. It can generalise a preliminary *main problem* that most of the semantics mentioned above and presented in Chapter 4 fail to overcome. It also help illustrate one

of the main claims of this thesis to formulate a semantics based on models rather than syntax to represent a propositional theory by means of ASP. A little-different original version of the coded story has been employed by Alferes et al. to point out problems with, what they call, *tautological rules*. However, the current version of the example also represents a problem for them, as shown in Section 4.2.

In order to better understand the example and before introducing formal definitions, the reader should read "$\neg$" as "*no evidence*" and "$\sim$" as "*not*". Finally, "$\leftarrow$" (respectively "$\leftrightarrow$") should be read as a *consequence relation*, similar to the one used in `Prolog`'s *if*: "`:-` ".

Consider the following scenario, first proposed by [Alferes et al., 2005][1] and here modified, describing some beliefs about the sky.

Example 1.1. *Suppose an agent who believes that* when *it is* day *it is not* night *and vice versa, and that there are* stars *when it is* night *and when there are no* clouds. *Finally, that at the current moment it is a fact that there are no* stars. *This simple story may be coded into* Π_1 *as follows:*

$$\begin{aligned} \Pi_1 = \{ day &\leftarrow \neg night \\ night &\leftarrow \neg day \\ stars &\leftarrow night, \neg cloudy \\ \sim stars & \quad \} \end{aligned}$$

whose unique answer set is $\{day, \sim stars\}$. *Later, the agent acquires new information stating that* stars *and* constls *(*constellations*) are the same thing, as coded in* Π_2. *As soon as the agent updates* Π_1 *with program*

$$\Pi_2 = \{stars \leftrightarrow constls\}$$

the expanded alphabet *of the two programs contains only one new* extra atom *with respect to* Π_1*: constls. As the model of* Π_2 *is obviously the empty answer set, constls is considered* synonym *of stars by means of* Π_2*, and thus the update should not change the* original beliefs.

If one carefully analysed this example, it would be easy to realise that the update does not really generate new information, and previous knowledge should remain unchanged. So there is a naive solution to the problem, consisting of a simple union of the two propositional theories in question, because the union remains consistent and does not change previous knowledge.

[1] Please note that the semantics presented in [Alferes et al., 2005] has no strong negation, and thus they use a syntax-dependent *"default" negation in heads!* to have a similar effect when updating with *tautological rules*, like $\{a \leftarrow a\}$, although in a peculiar way.

The problem with other approaches studied in Chapter 4 is that most of them are modifying the initial knowledge base justified by an apparent conflict between a *negative fact* of an old rule and its *positive counterpart* as a consequence of a new rule.

On the other hand, a few approaches can overcome this problem but still present shortcomings in some principles from the literature, like information loss, semantic justification and update characterisation. All of the problems are fully analysed in the chapters to come.

1.4.3 A Principle-based Approach to Represent Knowledge and Beliefs

By extending Kowalski's principle (i.e. his *equation* from Section 1.3), one of the claims of this thesis states that the process or *semantics* to change knowledge in a dynamic environment should be at a lower *epistemic level* of it, and constitute a *theory of belief change* rather than a specific procedure to achieve a particular change. As a result, the logic programming to formulate such a semantics should be declarative and implemented solvers may be allowed to use procedures that would illustrate *how to compute* it, mainly in its "user" interface —*modular interface*. In other words,

$$\text{Belief Change} = \text{logic} + \text{principles} + \text{control}$$

In order to solve the equation, the approaches proposed in this thesis shall depend on the *semantic contents* of logic programs, rather than the particular syntax they are written with. As a result, a semantics ought to meet two key *principles*: *Weak Irrelevance of Syntax* and *Strong Consistency*.

Intuitively, Strong Consistency states that supplementary rules like $\{a \leftrightarrow b\}$, should not result in any additional models, provided that a or b are atoms already in the underlying language. Such a property implies that the update should coincide with the union of the theories in question whenever the union is consistent. The property is a particular case of the well-known *Bordiga's principle*, introduced in Chapter 2. Weak Irrelevance of Syntax, on the other hand, says that one of two *logically-equivalent* theories updating a third one shall give the same result than updating with the other. This property is also a particular case of another well-known principle called *Dalal's principle of irrelevance of syntax*, introduced in the same Chapter 2. Further structural properties for updates of logic programs, as well as more general principles from the literature are also considered in this work.

Finally, as an important component of logic programming, this thesis provides online *solver prototypes* that support their declarative semantics and help close the gap between theory and practice. These labs of automatic testbeds make the semantics more accessible and opens a path with a solid component for further more-complex prototypes in management of *knowledge systems*.

1.5 Summary of Contributions

This section summarises the contributions of the research work that has been documented along this dissertation. In general, the contributions of this thesis may be grouped into two main streams: one that corresponds to the theoretical aspects of updating logic programs, and the other groups practical results to confirm claims from them. Next, a third group includes some work that may be relevant to further research. Finally, Section 9.2 includes a discussion about potential interest and impact that these contributions may have on the research community. Although they have an associated number, it does not necessarily reflect any other order of importance.

The theoretical side of this dissertation includes the following contributions.

1. a case study on how redundant updates jeopardise the integrity of most of existing semantics for updates of logic programs —Section 1.4.2 and Chapter 4 and Chapter 5.
2. a preliminary study on how *information loss* may be relevant in different scenarios for existing semantics for updates of logic programs —Section 4.3 and Chapter 5.
3. a general method of knowledge *relaxation* to *reasoning about inconsistent information* —Chapter 6.
4. a preliminary basic collection of structural properties to be satisfied by a reasonable semantics to override problems with *redundant information* —Section 6.2.
5. a study on how to deal with inconsistencies to perform both belief revision and updates —Section 7.3.1, Section 7.5 and Section 8.3.3.
6. a case-study to update sequences of logic programs that meet the proposed structural properties —Section 7.
7. a particular interpretation of AGM-postulates in ASP to deal with the non-monotonic nature of ASP—Section 8.3.4.
8. a characterisation of updates with preferences of both *cardinality* and *set-inclusion* criteria to satisfy particular requirements of *minimal change*, *conflict resolution* and persitency —Section 6.3.6, Section 7.6 and Section 8.4.
9. characterisation of semantics for updates with *Ordered-disjunctive Logic Programming* —Section 6.3.6.
10. characterisation of semantics for updates with weak constraints —Section 8.4.2
11. a particular interpretation of AGM-postulates in ASP and a semantics that satisfies five out of six postulates —Section 8.3.4.

The applied results from the theoretical side of this dissertation include the following contributions.

1. automatic methods to translate *abductive programs* into *weak-constraints programs*, which also may automate the process of finding their *generalised answer sets* —Section 8.4.2.

2. two main functional prototype solvers to compute updates transformed into weak-constraint programs —Section 7.6 and Section 8.4, with analysis of *complexity* —Section 8.4.3.

Some work potentially relevant results to further research include

1. a survey of the most relevant semantics for updates of logic programs —Chapter 4

2. a *frame of reference*, consisting of a series of *challenging examples* that might serve as a preliminary "*benchmark*" to assess intuitive behaviour with respect to redundant updates of logic programs —Chapter 5.

In this thesis I argue that there are important principles of logic-program updates that have not yet been observed in most of the cited proposals. For instance, the two fundamental properties described above: are *Weak Irrelevance of Syntax* (WIS, Section 6.2), which suggests that, *if one can update a theory τ by τ_1, the result should only depend upon the logical contents of τ_1, and not on the particular syntax used to express τ_1*. Secondly, a property of *Strong Consistency* (Section 6.2) states that supplementary rules of equivalence like $\{a \leftrightarrow b\}$, should not result in any additional model, provided that a and b are atoms already contained in the underlying language. As a result, this thesis is a progressive[1] proposal of a general semantics to overcome the problems of current semantics by following the most relevant postulates and principles in the literature in order to give more intuitive results and to provide the most general semantics as possible. Moreover, in this thesis I describe and provide online solver prototypes of the proposed formulations.

1.6 Publications

This thesis is based upon a number of papers that are diluted into the core chapters of this work, and they are shown below.

Chapter 4 uses most of [Guadarrama, 2007d].

Chapter 5 is a compilation of the counter-intuitive examples and others, shown mainly in [Guadarrama, 2007d].

Chapter 6 uses most of [Guadarrama et al., 2005] and [Guadarrama, 2008c].

Chapter 7 is built upon the results presented in [Guadarrama et al., 2006], [Guadarrama, 2007a] and [Guadarrama, 2009].

[1] With this word I mean that the results start from the basic concept up to a generalised formulation.

Chapter 8 uses most of [Guadarrama, 2007c], [Guadarrama, 2008a] and [Guadarrama, 2008b].

Appendix B is a compilation of software tools mainly reported in [Guadarrama et al., 2005], [Guadarrama, 2007a] and [Guadarrama, 2008b].

For he looked for a city which hath foundations, whose maker and builder is God.

HEBREWS 11,10

Chapter 2

Foundations

This chapter is the suggested *theoretical basis* of this thesis: *logics*, *postulates* and non-monotonic reasoning, which shall be recapitulated in subsequent chapters. It is a collection of particular principles that should be observed when representing knowledge evolution and they include *non-classical logics* that have a strong relation with logic programming and reasoning, postulates for *belief revision* and for *updates* that state specifications for knowledge evolution, as well as relevant particular *properties* for updating knowledge bases. Although logic may be and is often omitted in *logic programming*, the purpose of introducing them in this dissertation is to serve as a reference for subsequent chapters, to found the notion of *coherence* and *inference engines* and to emphasise the importance of a solid theoretical foundation both to knowledge principles and to *dynamic knowledge representation*. For example, they are main deductive bases and provide further properties of ASP, as seen in upcoming chapters, like two kinds of *negation* in the context of knowledge and beliefs (Section 3.2), as well as notion of *program equivalence* in logic programs —Section 3.3.

In particular, the axioms of *non-classical logic* include *intuitionistic logic*, a *three-valued logic* and *Nelson's logic*, as well as a collection of related principles that goes from the original AGM-postulates [Alchourrón et al., 1985], to diverse interpretations for particular purposes.

2.1 Logics

This section consists of several logics from the literature, as a major foundation to *reasoning*. It is assumed that the reader is familiar with classical logic and logic programming.

2.1.1 Intuitionistic Logic

To begin with, a good reference about logics and history may be found in the Encyclopædia Britannica [Safra and Yeshua, 2002] and in the Stanford Encyclopædia of

Philosophy, [Moschovakis, 1999], from which the following gentle gross introduction is compiled.

In 1907 the Dutch mathematician L.E.J. Brouwer questioned the principles of Classical Logic, that were considered untouchable until then [Brouwer, 1907]. In particular, the well-knonw *principle of the excluded middle* ($\mathfrak{A} \vee \neg\mathfrak{A}$) was criticised because it enforces that a truth-value is assigned without even *knowing* whether a constructive proof for $\mathfrak{A}$ (or its negation) can be obtained. In particular, Intuitionism takes only the "safe" principles from Classic Logic and gets rid of those for arbitrary undefined *"witnesses"* [Mints, 2000]. As a result, the axioms in Table 2.1.1 formalise the logic system, also known as *Heyting's Intuitionistic Logic*, or $\mathcal{H}_{\mathsf{I}}$ in short, and its language is defined as follows.

Definition 2.1 (Language $\mathcal{L}_{\mathcal{H}_{\mathsf{I}}}$ of *intuitionistic logic*)**.** *The language $\mathcal{L}_{\mathcal{H}_{\mathsf{I}}}$ is defined from propositional logic with propositional symbols ($p_0, p_1, \dots$); binary connectives ($\wedge, \vee, \supset, \bot, \top$); auxiliary symbols: "(" , ")" (parentheses); $\bot$ is* absurdity constant *and $\top$ is* true constant*. The propositional symbols are also called* atoms *or* atomic propositions*. Formulae and theories are defined as usual in logic. The formula $\neg\mathfrak{A}$ is introduced as an abbreviation of $\mathfrak{A} \supset \bot$.*

Equivalence $\equiv$, defined as $\mathfrak{A} \equiv \mathfrak{B}$, is shorthand for $\mathfrak{A} \supset \mathfrak{B} \wedge \mathfrak{B} \supset \mathfrak{A}$. In order to distinguish such relation from other notions of equivalence in upcoming chapters, it shall also be denoted as "$\equiv_{\mathcal{H}_{\mathsf{I}}}$". The set of axioms consists of A1-$\mathcal{H}_{\mathsf{I}}$ to A10-$\mathcal{H}_{\mathsf{I}}$ on Table 2.1.1.

A1-$\mathcal{H}_{\mathsf{I}}$: $\mathfrak{A} \supset (\mathfrak{B} \supset \mathfrak{A})$

A2-$\mathcal{H}_{\mathsf{I}}$: $(\mathfrak{A} \supset \mathfrak{B}) \supset ((\mathfrak{A} \supset (\mathfrak{B} \supset \mathfrak{C})) \supset (\mathfrak{A} \supset \mathfrak{C}))$

A3-$\mathcal{H}_{\mathsf{I}}$: $\mathfrak{A} \supset (\mathfrak{B} \supset \mathfrak{A} \wedge \mathfrak{B})$

A4-$\mathcal{H}_{\mathsf{I}}$: $\mathfrak{A} \wedge \mathfrak{B} \supset \mathfrak{A}$

A5-$\mathcal{H}_{\mathsf{I}}$: $\mathfrak{A} \wedge \mathfrak{B} \supset \mathfrak{B}$

A6-$\mathcal{H}_{\mathsf{I}}$: $\mathfrak{A} \supset \mathfrak{A} \vee \mathfrak{B}$

A7-$\mathcal{H}_{\mathsf{I}}$: $\mathfrak{B} \supset \mathfrak{A} \vee \mathfrak{B}$

A8-$\mathcal{H}_{\mathsf{I}}$: $(\mathfrak{A} \supset \mathfrak{C}) \supset ((\mathfrak{B} \supset \mathfrak{C}) \supset (\mathfrak{A} \vee \mathfrak{B} \supset \mathfrak{C}))$

A9-$\mathcal{H}_{\mathsf{I}}$: $(\mathfrak{A} \supset \mathfrak{B}) \supset ((\mathfrak{A} \supset \neg\mathfrak{B}) \supset \neg\mathfrak{A})$

A10-$\mathcal{H}_{\mathsf{I}}$: $\neg\mathfrak{A} \supset (\mathfrak{A} \supset \mathfrak{B})$

Table 2.1: Intuitionistic-logic Heytig's Axioms

In contrast, the following axioms from *classical logic* do not hold:

$$\neg\neg\mathfrak{A} \supset \mathfrak{A} \tag{2.1}$$
$$\mathfrak{A} \vee \neg\mathfrak{A} \tag{2.2}$$
$$\neg(\mathfrak{A} \vee \mathfrak{B}) \equiv \neg\mathfrak{A} \wedge \neg\mathfrak{B} \tag{2.3}$$

As in classical logic, *Modus Ponens* is the only inference rule.

The expression $\vdash_{\mathcal{K}} \mathfrak{A}$ denotes the standard derivation of $\mathfrak{A}$ from the axioms in system $\mathcal{K}$ using its axioms and derivation rules. In other words, $\mathfrak{A}$ is *provable* in system $\mathcal{K}$. Accordingly, $\mathfrak{A} \vdash_{\mathcal{K}} \mathfrak{B}$ denotes $\vdash_{\mathcal{K}} \mathfrak{A} \supset \mathfrak{B}$. As a result, if Γ is a set of formulae, $\Gamma \vdash_{\mathcal{K}} \mathfrak{A}$ means that from Γ the formula $\mathfrak{A}$ can be proved (in system $\mathcal{K}$).

Because Stable Models Semantics (SM) can be seen as Answer Set Programming with one *negation*, intuitionistic logic proves to be useful in characterising SM, as shown in Section 3.2, and also to help construct the following logic systems with one more negation and more than two *truth values*.

2.1.2 Multi-Valued Logic

Non-intuitionist logics may be defined in terms of truth-values and evaluation functions. Gödel's logic $\mathcal{G}_i$ is a generalisation of the well-know *truth tables* and its *interpretation* or *model* is defined as a function $I : \mathcal{L}_{\mathcal{G}_i} \longmapsto \{0, 1, \ldots, i-1\}$ (where $\mathcal{L}_{\mathcal{G}_i}$ is the set containing the language atoms), evaluated on logic formulae as follows:

$I(\mathfrak{A} \supset \mathfrak{B}) = i - 1$ if $I(\mathfrak{A}) \leq I(\mathfrak{B})$, and $I(\mathfrak{A} \supset \mathfrak{B}) = I(\mathfrak{B})$ otherwise.

$I(\mathfrak{A} \vee \mathfrak{B}) = \max(I(\mathfrak{A}), I(\mathfrak{B}))$.

$I(\mathfrak{A} \wedge \mathfrak{B}) = \min(I(\mathfrak{A}), I(\mathfrak{B}))$.

$I(\neg\mathfrak{A}) = 0$ if $I(\mathfrak{A}) > 0$ and $I(\neg\mathfrak{A}) = i - 1$ if $I(\mathfrak{A}) = 0$.

$I(\top) = i - 1$ and $I(\bot) = 0$.

Particularly in this work, $i = 3$ shall be enough to define $\mathcal{H}_{\mathsf{HT}}$-logic, that is to say, $\mathcal{G}_3$. It is also worth noticing that $\mathcal{G}_2$ corresponds to the axioms of *classical logic*. In addition, $\mathcal{H}_{\mathsf{HT}}$-logic may also be defined in terms of the intuitionist axioms A1-$\mathcal{H}_{\mathsf{I}}$–A10-$\mathcal{H}_{\mathsf{I}}$ from Table 2.1.1 —see [Moschovakis, 1999]— and the axiom A1-$\mathcal{H}_{\mathsf{HT}}$ [Pearce, 1999a][1]:

A1-$\mathcal{H}_{\mathsf{HT}}$:

$$(\mathfrak{A} \supset \neg\mathfrak{B}) \supset (((\mathfrak{B} \supset \mathfrak{A}) \supset \mathfrak{B}) \supset \mathfrak{B}).$$

[1] Note that [Lifschitz et al., 2001] state that they can obtain $\mathcal{H}_{\mathsf{HT}}$-logic from $\mathcal{H}_{\mathsf{I}}$ by adding the axiom $\mathfrak{A} \vee (\mathfrak{A} \supset \mathfrak{B}) \vee \neg\mathfrak{B}$. Proving equivalence between the two statements lies beyond the intended focus of this dissertation.

As said above, logics may be defined in terms of truth-values and evaluation functions. As a result, Gödel's logic $\mathcal{G}_3$ is defined as a three-valued logic, with values in $\{0, 1, 2\}$ where 2 is the designated value, with the following *evaluation function* $f : \mathcal{L}_{\mathcal{G}_3} \longmapsto \{0, 1, 2\}$ as follows

$f(\mathfrak{A} \supset \mathfrak{B}) = 2$ if $f(\mathfrak{A}) \leq f(\mathfrak{B})$, and $f(\mathfrak{A} \supset \mathfrak{B}) = f(\mathfrak{B})$ otherwise.

$f(\mathfrak{A} \vee \mathfrak{B}) = \max(f(\mathfrak{A}), f(\mathfrak{B}))$.

$f(\mathfrak{A} \wedge \mathfrak{B}) = \min(f(\mathfrak{A}), f(\mathfrak{B}))$.

$f(\neg\mathfrak{A}) = 0$ if $f(\mathfrak{A}) > 0$ and $f(\neg\mathfrak{A}) = 2$ if $f(\mathfrak{A}) = 0$.

$f(\top) = 2$ and $f(\bot) = 0$.

An interpretation in such three-valued logics is the function that assigns to each atom in $\mathcal{L}_{\mathcal{G}_i}$ a value from $\{0, 1, 2\}$. The interpretation of an arbitrary formula is obtained propagating the valuation of each connective as defined above. An interpretation is said to be *definite* if it assigns only values 0 or 2, and *indefinite* if some intermediate value is assigned to an atom. In addition, given an interpretation I, we write $I(p)$ to denote the value that I assigns to the atom p.

For a given interpretation I and a formula ϕ we say that I is a $\mathcal{G}_3$ *model* of ϕ (or I models ϕ) if $I(\phi) = 2$. Of course, we extend this definition as usual to a theory (set of formulas). Last, a *tautology* is a formula that evaluates to 2 for every possible interpretation.

An interesting application of this logic is in alternative *related work* like *paraconsistent systems*, as suggested in [Guadarrama et al., 2002; Guadarrama and Osorio, 2002, 2003; Osorio et al., 2005]. On the other hand, in this particular thesis it is useful to understand a second kind of negation and to introduce Nelson's logics.

2.1.3 Nelson's Logic and $\mathcal{N}_2$-logic

Another *non-classical logic*, foundation of this thesis, is $\mathcal{N}$-*logic*, which is an extension to *intuitionistic logic* by expanding $\mathcal{L}_{\mathcal{H}_\mathsf{I}}$ with a second kind of negation, "$\sim$", known in the literature as *strong negation* for $\vdash_{\mathcal{N}} \sim\mathfrak{A} \supset \neg\mathfrak{A}$, and by adding to $\mathcal{H}_\mathsf{I}$ the axioms in Table 2.2. Other names for this second kind of negation sketched before in Section 1.3 are *explicit negation*, *classical negation* or also *constructible falsity*. In this dissertation, such negation shall be represented as "$\sim$".

Finally, $\mathcal{N}_2$-*logic* consists of axioms

A1-$\mathcal{N}_2$**:** $\mathfrak{A} \vee (\mathfrak{A} \supset \mathfrak{B}) \vee \neg\mathfrak{B}$

as well as $\mathcal{N}$*'s axioms* [Pearce, 1999a]. Logic $\mathcal{N}_2$ will prove to be a main *deduction system* to ASP in Section 3.2 and Section 3.3, as $\mathcal{H}_{\mathsf{HT}}$ and $\mathcal{H}_\mathsf{I}$ are to SM.

An intuitive meaning of *strong negation* "$\sim$" and the more classical *logic-programming negation* "$\neg$" is that $\neg p$ can also be denoted as $\bot \leftarrow p$, i.e., "$\neg$" means that it is *believed*

A1-$\mathcal{N}$: $\sim(\mathfrak{A} \wedge \mathfrak{B}) \equiv (\sim\mathfrak{A} \vee \sim\mathfrak{B})$

A2-$\mathcal{N}$: $\sim(\mathfrak{A} \vee \mathfrak{B}) \equiv (\sim\mathfrak{A} \wedge \sim\mathfrak{B})$

A3-$\mathcal{N}$: $\sim(\mathfrak{A} \supset \mathfrak{B}) \equiv (\mathfrak{A} \wedge \sim\mathfrak{B})$

A4-$\mathcal{N}$: $\sim\sim\mathfrak{A} \equiv \mathfrak{A}$

A5-$\mathcal{N}$: $\sim\neg\mathfrak{A} \equiv \mathfrak{A}$

A6-$\mathcal{N}$: $\sim\mathfrak{A} \supset \neg\mathfrak{A}$, for an atomic $\mathfrak{A}$.

A1-$\mathcal{H}_\mathsf{I}$–A10-$\mathcal{H}_\mathsf{I}$: inclusive.

Table 2.2: $\mathcal{N}$-axioms, including $\mathcal{H}_\mathsf{I}$ axioms from Table 2.1.1

that there is *no evidence* from p: p is not true by default or not *provable*. In contrast, $\sim p$ is stated when it is *known* that p does not exist, is false or does not happen.

In the particular language of ASP introduced later in Section 3.2, the *strong negation* symbol is allowed to occur at the atomic level only. Namely, one may have expressions like $\sim a$ but never expressions like $\sim\neg a$ as in the axiom A5-$\mathcal{N}$, which is an extra axiom with respect to $\mathcal{G}_3$.

To sum up, $\mathcal{N}$-logic is an extension of $\mathcal{H}_\mathsf{I}$ for it includes axioms A1-$\mathcal{H}_\mathsf{I}$–A10-$\mathcal{H}_\mathsf{I}$, axioms A1-$\mathcal{N}$–A6-$\mathcal{N}$ and strong negation; while $\mathcal{N}_2$ is an extension of $\mathcal{H}_\mathsf{I}$ with axiom A1-$\mathcal{N}_2$ and $\mathcal{N}$ axioms. On the other hand, $\mathcal{H}_\mathsf{HT}$-logic is an extension of $\mathcal{H}_\mathsf{I}$-logic for the axiom A1-$\mathcal{H}_\mathsf{HT}$. Finally, $\mathcal{N}$-logic is also an extension of $\mathcal{H}_\mathsf{HT}$, but their differences lie beyond the intended focus of this research[1]. The focus of this research shall be on ASP, where $\mathcal{N}_2$-logic shall prove to have more advantages for it, as discussed in upcoming sections, like Section 3.2.

Although logics are meant to manipulate *theories* (sets of formulas closed under a logical inference), the logics themselves do not specify any particular structure for such theories or where they are to be held. In following sections, include frameworks to realise it.

Up to now, this is some of the necessary framework to *reasoning*, that underpins decades of research and resulting semantics to solve problems in a *static* and/or non-monotonic setting. However, one of the main goals of this thesis is to correctly represent knowledge, which is the endeavour of upcoming sections on research of principles and proposals to realise it.

2.2 Change and Belief Representation

Once the machinery to *reason about knowledge* has been established, a following evident need is how to represent knowledge. Although the logics introduced in Section 2.1 are

[1]For further details, refer to [Pearce, 1999a].

meant to manipulate *logical theories* (sets of formulas closed under a logical inference), the logics themselves do not specify any particular structure for such theories or where they are to be held. An obvious general way to group them is to associate the theories with sets of beliefs, what is commonly known in the literature as *knowledge sets* or *belief sets*.

In the context of this research work of *evolution of knowledge* (thus incomplete), let us adopt the same Gärdenfors and Makinson's stance: a *theory* or a *belief set* is a *partial description* of the environment that represent commitments. On the other hand, *beliefs* are to be partially represented as *sentences* in a formal language, $\mathcal{L}$.

In particular, Alchourrón et al. define a *consequence operation* "Cn" as a function from *sets of propositions* to sets of propositions that satisfy three conditions: Given two sets of propositions, Γ_1, Γ_2, in a language $\mathcal{L}$,

inclusion: $\Gamma_1 \subseteq \mathsf{Cn}(\Gamma_1)$.

iteration: $\mathsf{Cn}(\Gamma_1) = \mathsf{Cn}(\mathsf{Cn}(\Gamma_1))$.

monotony: $\mathsf{Cn}(\Gamma_1) \subseteq \mathsf{Cn}(\Gamma_2)$ when $\Gamma_1 \subseteq \Gamma_2$.

In addition, $\Gamma_1 \vdash \psi$ is an abbreviation for $\psi \in \mathsf{Cn}(\Gamma_1)$ that means Γ_1 *logically entails* ψ. Accordingly, $\mathsf{Cn}(\Gamma_1) = \{y \mid \Gamma_1 \vdash y\}$ and a theory is a set of propositions A closed under Cn. In symbols, $A = \mathsf{Cn}(B)$ for some set of propositions, B.

It is assumed that Cn includes classical implication and propositional language connectives and constants. Moreover, an *inconsistent* theory Γ means that $\Gamma \vdash \phi$ for every sentence ϕ of the language $\mathcal{L}$, denoted as $\Gamma_\perp$.

2.3 Belief Revision

The problem of *belief revision* comes up when integrating different sources of information into a current belief system, with a minimal impact on such change. In addition, the system must be kept *consistent*. There are at least two situations in which the information may be inconsistent: that new information contradicts previous one, and that the integrity of current information need be restored. The former may have to do with updates, depending on the "openness" of the environment, as discussed in Section 1.2.3, and recapped in Section 2.4 and upcoming chapters. On the other hand, the other situation has to do just with belief revision, which is the focus of this section.

After years of research, Alchourrón et al. formulated a set of properties a belief revision mechanism should meet, proposed into a series of eight postulates that are described in Table 2.3.

As sketched earlier in Section 1.2.3, they identified three main operations known as *expansion*, *revision* and *contraction*, here denoted with operators $\uplus$, $\dot{+}$ and $\ominus$, respectively. In particular, given a theory Γ and a proposition ϕ, *expansion* consists in adding the latter to Γ without any concern about its conflict with other sentences. In contrast, *contraction* removes ϕ from Γ so that it is no longer implied by the resulting set of

($\mathsf{R}\dot{+}\mathbf{1}$) **Closure**: $\Gamma \dot{+} \phi = \mathsf{Cn}(\Gamma \dot{+} \phi)$ is a *theory*

($\mathsf{R}\dot{+}\mathbf{2}$) **Success**: $\phi \in \Gamma \dot{+} \phi$

($\mathsf{R}\dot{+}\mathbf{3}$) **Inclusion**: $\Gamma \dot{+} \phi \subseteq \mathsf{Cn}(\Gamma \cup \{\phi\})$

($\mathsf{R}\dot{+}\mathbf{4}$) **Vacuity**: If $\neg\phi \notin \Gamma$, then $\mathsf{Cn}(\Gamma \cup \{\phi\}) \subseteq \Gamma \dot{+} \phi$

($\mathsf{R}\dot{+}\mathbf{5}$) **Consistency**: $\Gamma \dot{+} \phi = \Gamma_\perp$ iff $\vdash \neg\phi$

($\mathsf{R}\dot{+}\mathbf{6}$) **Extensionality**: If $\vdash \phi \leftrightarrow \psi$, then $\Gamma \dot{+} \phi = \Gamma \dot{+} \psi$

($\mathsf{R}\dot{+}\mathbf{7}$) **Super-expansion**: $\Gamma \dot{+} (\phi \wedge \psi) \subseteq \mathsf{Cn}(\Gamma \dot{+} \phi \cup \{\psi\})$

($\mathsf{R}\dot{+}\mathbf{8}$) **Sub-expansion**: If $\neg\psi \notin \Gamma \dot{+} \phi$, then $\mathsf{Cn}(\Gamma \dot{+} \phi \cup \{\psi\}) \subseteq \Gamma \dot{+} (\phi \wedge \psi)$

Table 2.3: AGM-postulates

sentences, together with some more propositions to make the set closed under logical consequences. Finally, *revision* is an expansion of a new sentence ϕ and contraction of its possible inconsistent sentences from Γ.

In terms of logical closure, expansion operator "$\uplus$" is defined as

$$\Gamma \uplus \phi = \{\psi \mid \Gamma \cup \{\phi\} \vdash \psi\} = \mathsf{Cn}(\Gamma \cup \{\phi\})$$

With two out of the three operations, Alchourrón et al. formulated a set of *postulates* that specify changes to knowledge base, better known as *AGM-postulates*. Table 2.3 shows a version of them in my own notation.

However, a problem in computer science is that belief sets may be *infinite* and unspecified. As a result, a more practical way to represent and manipulate theories seems to be arbitrary *sets of propositional sentences* known as *knowledge bases* rather than belief sets; and for the particular application of Artificial Intelligence, *finite knowledge bases* [Katsuno and Mendelzon, 1991b] without making any distinction between *knowledge and beliefs*. Formally, $\mathcal{K}$ is a *base for a belief set* Γ if and only if $\mathcal{K}$ is a finite subset of Γ and $\mathsf{Cn}(\mathcal{K}) = \Gamma$.

Accordingly, Katsuno and Mendelzon redefined the AGM-postulates in terms of *knowledge bases* by representing any belief set Γ with some *propositional formula* ϕ such that $\Gamma = \{\psi \mid \phi \vdash \psi\}$. By satisfying four of their new postulates —($\mathsf{R}*\mathbf{1}$)–($\mathsf{R}*\mathbf{4}$)— they prove to satisfy six of the AGM-postulates —($\mathsf{R}\dot{+}\mathbf{1}$)–($\mathsf{R}\dot{+}\mathbf{6}$). In addition, the remaining two postulates ($\mathsf{R}*\mathbf{5}$)–($\mathsf{R}*\mathbf{6}$) are equivalent to ($\mathsf{R}\dot{+}\mathbf{7}$) and ($\mathsf{R}\dot{+}\mathbf{8}$), respectively. Their paraphrase with $*$-operator is shown in Table 2.4, which gave a basis to further postulates to differentiate belief update from belief revision introduced later in Section 2.4.

Finally, Darwiche and Pearl put forward yet another interpretation of postulates ($\mathsf{R}*\mathbf{1}$)–($\mathsf{R}*\mathbf{6}$), by means of two proposals: that a belief revision operation should be

($\mathsf{R}*\mathbf{1}$) $\phi * \chi$ implies χ.

($\mathsf{R}*\mathbf{2}$) If $\phi \wedge \chi$ is satisfiable, then $\phi * \chi \leftrightarrow \phi \wedge \chi$.

($\mathsf{R}*\mathbf{3}$) If χ is satisfiable, then $\phi * \chi$ is also satisfiable.

($\mathsf{R}*\mathbf{4}$) If $\phi_1 \leftrightarrow \phi_2$ and $\chi_1 \leftrightarrow \chi_2$ then $\phi_1 * \chi_1 \leftrightarrow \phi_2 * \chi_2$.

($\mathsf{R}*\mathbf{5}$) $(\phi * \chi) \wedge \mu$ implies $\phi * (\chi \wedge \mu)$.

($\mathsf{R}*\mathbf{6}$) If $(\phi * \chi) \wedge \mu$ is satisfiable, then $\phi * (\chi \wedge \mu)$ implies $(\phi * \chi) \wedge \mu$.

Table 2.4: KM-postulates —Katsuno and Mendelzon's AGM-interpretation

($\mathsf{R}\circ\mathbf{1}$) $Bel(\mathcal{E} \circ \chi)$ implies χ.

($\mathsf{R}\circ\mathbf{2}$) If $Bel(\mathcal{E}) \wedge \chi$ is satisfiable, then $Bel(\mathcal{E} \circ \chi) \equiv Bel(\mathcal{E}) \wedge \chi$.

($\mathsf{R}\circ\mathbf{3}$) If χ is satisfiable, then $Bel(\mathcal{E} \circ \chi)$ is also satisfiable.

($\mathsf{R}\circ\mathbf{4}$) If $\mathcal{E}_1 = \mathcal{E}_2$ and $\chi_1 \equiv \chi_2$ then $Bel(\mathcal{E}_1 \circ \chi_1) \equiv Bel(\mathcal{E}_2 \circ \chi_2)$.

($\mathsf{R}\circ\mathbf{5}$) $Bel(\mathcal{E} \circ \chi) \wedge \mu$ implies $Bel(\mathcal{E} \circ (\chi \wedge \mu))$.

($\mathsf{R}\circ\mathbf{6}$) If $Bel(\mathcal{E} \circ \chi) \wedge \mu$ is satisfiable, then $Bel(\mathcal{E} \circ (\chi \wedge \mu))$ implies $Bel(\mathcal{E} \circ \chi) \wedge \mu$.

Table 2.5: KM′-postulates —Darwiche and Pearl's KM paraphrase

performed on an *epistemic state* rather than on a knowledge base; and that postulate ($\mathsf{R}*\mathsf{4}$) should be more cautious by requiring identical epistemic states (ref. sets) rather than equivalent belief sets (ref. states). They proposed that each epistemic state should include the particular specifications to perform belief revision on the beliefs themselves, as well as the necessary information to reason with such beliefs.

Accordingly, each *epistemic state* $\mathcal{E}$ has an associated belief set $Bel(\mathcal{E})$ that can itself be represented by some propositional formula ϕ that is a knowledge base for it when it –$Bel(\mathcal{E})$– is finite, as Katsuno and Mendelzon proposed earlier. As a result, a belief base $\mathcal{K} \subseteq Bel(\mathcal{E})$, such that $Bel(\mathcal{E}) = \mathsf{Cn}(\mathcal{K})$. However, each (ref. equivalent) belief set(s) may have more than one epistemic state, which means that $\mathcal{E}$ and $Bel(\mathcal{E})$ do not necessary have the same language. In other words, $Bel(\mathcal{E}) = \{\psi \mid \phi \vdash \psi\}$ where ϕ is a base for a finite belief set of $\mathcal{E}$, and $Bel(\mathcal{E}) \vdash \psi$. On the other hand, it is possible to have more than one epistemic state with equivalent (ref. the same) belief sets.

The proposed interpretation to KM-postulates in Table 2.4 are shown in Table 2.5 as KM′-postulates, where the new revision operator "$\circ$" performs over an *epistemic state* updated with a *propositional sentence*, which results in a new epistemic state. That is to say, $\mathcal{E} \circ \chi$ is an *epistemic state*.

Just after these rephrases over rephrases of the original AGM-postulates, a new

(**U** $\circledast$ **1**) $\phi \circledast \chi$ implies χ.

(**U** $\circledast$ **2**) If ϕ implies χ then $\phi \circledast \chi$ is equivalent to ϕ.

(**U** $\circledast$ **3**) If both ϕ and χ are satisfiable then $\phi \circledast \chi$ is also satisfiable.

(**U** $\circledast$ **4**) If $\phi_1 \leftrightarrow \phi_2$ and $\chi_1 \leftrightarrow \chi_2$ then $\phi_1 \circledast \chi_1 \leftrightarrow \phi_2 \circledast \chi_2$.

(**U** $\circledast$ **5**) $(\phi \circledast \chi) \wedge \phi$ implies $\phi \circledast (\chi \wedge \phi)$.

(**U** $\circledast$ **6**) If $\phi \circledast \chi_1$ implies χ_2 and $\phi \circledast \chi_2$ implies χ_1 then $\phi \circledast \chi_1 \leftrightarrow \phi \circledast \chi_2$

(**U** $\circledast$ **7**) If ϕ is complete then $(\phi \circledast \chi_1) \wedge (\phi \circledast \chi_2)$ implies $\phi \circledast (\chi_1 \vee \chi_2)$.

(**U** $\circledast$ **8**) $(\phi_1 \vee \phi_2) \circledast \chi \leftrightarrow (\phi_1 \circledast \chi) \vee (\phi_2 \circledast \chi)$.

where a knowledge base ϕ is *complete* iff, for each atom A, either $A \in \phi$ or $\neg A \in \phi$.

Table 2.6: Belief Update on Knowledge Bases, by Katsuno and Mendelzon.

theory of change would question them again, stating that belief-revision operations would not solve certain specific problems. That is matter of the following section.

2.4 Belief Update

As one of the major and traditional topics of Artificial Intelligence over the last years, knowledge representation and reasoning has proved to be a strong theoretical framework for Logic Programming to manage knowledge bases. As a result, this particular topic has become more widely applied in the administration knowledge bases of intelligent (rational) agents, particularly when considering an agent's incomplete knowledge in a changing environment. This area of research is known in the literature as *belief updates* and has inspired numerous proposals to update logic programs, as presented in Chapter 4.

As discussed in Section 2.3, Katsuno and Mendelzon redefined the AGM-postulates in terms of knowledge bases rather than belief sets. Having done that, they also proposed to make a *difference between belief revision and belief updates*, arguing that belief revision is not adequate to solve some particular problems [Katsuno and Mendelzon, 1991a].

Specifically, they state that an *update to a knowledge base* brings it up-to-date when the environment described by it changes. *belief revision*, on the other hand, incorporates new information to the knowledge base from the modelled environment that never changes. As a result, Katsuno and Mendelzon came off with a set of eight new *postulates for belief updates* that a generic update operator "$\circledast$" ought to satisfy, as shown in Table2.6.

Accordingly, Katsuno and Mendelzon state that each revision postulate (R$*$1)–(R$*$5) has a corresponding one in the other system. However, one of the main differences

between updating and revising a knowledge base lies in postulate (U ⊛ 2), which says that an update to a knowledge base with a derived proposition does not alter the knowledge base. This means that both the knowledge base and the update have to be consistent to guarantee a consistent result, as stated in (U⊛3). Otherwise, the resulting knowledge base shall be inconsistent no matter what the update is, as formally expressed in Lemma 2.1:

Lemma 2.1 (Updated inconsistency [Katsuno and Mendelzon, 1991a])**.** *If an update operator ⊛ satisfies (U⊛2), and ψ is* inconsistent*, then $\psi \circledast \chi$ is inconsistent for any χ.*

On the other hand, postulate (R∗3) states that the revision of a *knowledge base* is always consistent, provided that the *revising sentence* is *consistent* too. As a consequence, there is an implicit *Consistency Restoration* by a revision function in the postulates, for an original knowledge base. Postulate (U ⊛ 3), in contrast, is more precise and requires that both the original knowledge base and the update are consistent so that the resulting knowledge base is also consistent.

Moreover, they add postulates (U ⊛ 6)–(U ⊛ 8) instead of (R ∗ 6). According to Katsuno and Mendelzon, (U⊛7) states that some *possible world* resulting from updating a complete knowledge base with an update and also resulting from another update, such a possible world must also result from updating the knowledge base with the disjunction of the two updates. On the other hand, (U⊛8) states that updating each *possible world* of a knowledge base has independent updates.

Finally, let us introduce an interpretation of postulates (U ⊛ 1)–(U ⊛ 8) in terms of belief states, where each *epistemic state* $\mathcal{E}$ has an associated belief set $Bel(\mathcal{E})$, which is itself a proposition ϕ, as stated earlier in Section 2.3. Moreover, "$\diamond$" is a generic update operator over an *epistemic state* updated with a propositional sentence, which results in a new epistemic state. That is to say, $\mathcal{E} \diamond \chi$ is an *epistemic state*. The set of redefined postulates from Eiter et al. are shown in Table 2.7.

2.5 Conclusion for Chapter 2

This chapter consists of a progressive collection from the literature of logic axioms as a major theoretical foundation to reasoning and knowledge representation, as well as belief postulates and principles that specify regulations to modify knowledge and to reason about changes in logical settings.

The main stream has its origin in representation of theories as sets of formulas closed under a logical inference. Such sets of formulas are associated with *belief sets* that are a partial description of the world. Out of these groups, one can define basic properties as iteration, inclusion and monotony.

As a partial description of the world, the evolution of such groups of formulas makes them prone to run across contradictory information, which is to be incorporated intro the group of formulas to draw consistent conclusions. *belief revision postulates* are specifications of minimal changes to those structures and the first formulation is the

($\mathbf{U} \diamond \mathbf{1}$) $\phi \in Bel(\mathcal{E} \diamond \phi)$.

($\mathbf{U} \diamond \mathbf{2}$) $\phi \in Bel(\mathcal{E})$ implies $Bel(\mathcal{E} \diamond \phi) = Bel(\mathcal{E})$.

($\mathbf{U} \diamond \mathbf{3}$) If $Bel(\mathcal{E})$ is consistent and ϕ is satisfiable, then $Bel(\mathcal{E} \diamond \phi)$ is consistent.

($\mathbf{U} \diamond \mathbf{4}$) If $Bel(\mathcal{E}) = Bel(\mathcal{E}')$ and $\phi \equiv \psi$, then $Bel(\mathcal{E} \diamond \phi) = Bel(\mathcal{E}' \diamond \psi)$.

($\mathbf{U} \diamond \mathbf{5}$) $Bel(\mathcal{E} \diamond (\phi \wedge \psi)) \subseteq Bel((\mathcal{E} \diamond \phi) \uplus \psi)$.

($\mathbf{U} \diamond \mathbf{6}$) If $\phi \in Bel(\mathcal{E} \diamond \psi)$ and $\psi \in Bel(\mathcal{E} \diamond \phi)$, then $Bel(\mathcal{E} \diamond \phi) = Bel(\mathcal{E} \diamond \psi)$.

($\mathbf{U} \diamond \mathbf{7}$) If $Bel(\mathcal{E})$ is complete, then $Bel(\mathcal{E} \diamond (\phi \vee \phi')) \subseteq Bel(\mathcal{E} \diamond \phi) \wedge Bel(\mathcal{E} \diamond \phi')$

($\mathbf{U} \diamond \mathbf{8}$) $Bel((\mathcal{E} \vee \mathcal{E}') \diamond \phi) = Bel((\mathcal{E} \diamond \phi) \vee (\mathcal{E}' \diamond \phi))$.

where a belief set K is *complete* iff, for each atom A, either $A \in K$ or $\neg A \in K$.

Table 2.7: Belief Update Postulates for Epistemic States

AGM-postulates. The structures on which they specify changes are *belief sets* closed under a logical consequence, by means of three operations: expansion, revision and contraction.

In order to make such a framework appropriate to computer science, Katsuno and Mendelzon redefined the AGM-postulates in terms of finite belief sets not logically closed, known as *knowledge bases*, which of course, can have an associated belief set closed under a logical consequence. These KM-postulates also included an equivalent abbreviation from eight to six postulates. At the same time, they postulated new specifications to establish subtle differences between belief revision and *belief update*, also in terms of knowledge bases. With the new postulates, they argued that belief update represent beliefs in a changing environment, while belief revision in a static one, with the corresponding consequences.

Although the main proposed semantics in this thesis are called "semantics for updates of logic programs" for historical and practical reasons, the main focus, according to Katsuno and Mendelzon terms, shall be on belief revision, as further discussed in subsequent sections starting with Section 7.5.

Finally, Darwiche and Pearl made yet another refinement to KM-postulates in terms of *epistemic states* rather than belief bases. One of the purposes of such tune-up is to encode into an epistemic state, both the necessary information to reason and the particular specifications to further belief revision.

To sum up, a general difference between belief revision and updates is the changing environment, and an agent should both *correct* its own *misconceptions* of it (*belief revision*) and *suppose* that the environment exceptionally changes. As a result, a useful semantics to model such changes is one that performs as many principles as possible, from belief revision and updates of epistemic states, including the necessary machinery to perform coherent reasoning.

Readability is more important than parsimony.

Russell & Norvig

Chapter 3

Preliminaries

This chapter is a global compilation of a standard notation employed along this thesis, from *logic programming*, *Answer Set Programming* (ASP) and *Generalised Answer Sets* or GAS. In particular, ASP is a *knowledge-representation* semantics that has several classes of programs from which this chapter comprises *extended disjunctive logic programs* as the main core of this work. Finally, GAS is a semantics on ASP that shall prove to be helpful in preferring models to characterise updates of logic programs. The reader is expected to be familiar with definitions and basic notions of logic —see for example Chapter 1 of [Shankar, 1997] or [Lloyd, 1987]. Additionally, this chapter includes standard notation of complexity classes as in [Johnson, 1990].

3.1 Logic Programming and Answer Sets

Up to Chapter 2, I have considered the language of *classical logic* and introduced some logic systems with their respective axioms, which are a very abstract foundation to this thesis. The same chapter also includes some general logical principles that lead to a *correct belief-change*. Such strong theoretical bases, however, may not compute the changes in the knowledge bases described by their sole principles, and need to be integrated into another *operational framework* that inherits the desired properties to perform *belief changes*.

3.2 Stable Models and Answer Sets

One foundation of this thesis is *Answer Set Programming*, ASP [see Gelfond and Lifschitz, 1988; Lifschitz and Woo, 1992], *characterised* in some non-classical logics introduced in Chapter 2 due to [Osorio et al., 2004; Pearce, 1999a,b], with a long background and suitability to represent *non-monotonic knowledge*. Its main applications in *problem solutions* range from typical AI *toy examples* to yet-preliminary *agent prototypes* and *planning settings*. Other name to identify this semantics from the literature is *Stable*

Model Semantics or simply SM for its name in the original paper [Gelfond and Lifschitz, 1988].

Characterising ASP with a logic means that a *propositional theory* represented by a program over the non-monotonic answer-sets language can be inferred in terms of a monotonic logic [Osorio et al., 2004]. In other words, there are particular logics that are *deductive bases* of ASP inferences —i.e. *Here-and-There Logic* and $\mathcal{N}_2$*-logic*— [Pearce, 1999a]. Accordingly, one may take advantage of some *inference features* of a particular logic that are not so obvious in ASP. For example, one may transform *nested expressions* in logic to its equivalent in ASP or one may check *strong equivalence* of logic programs by proving equivalence in a particular logic system, as shown in upcoming sections.

The following formalism gives the description of ASP, which is identified with other names like *Stable Logic Programming* or *Stable Model Semantics* and A-Prolog. Its formal language and some more notation are introduced as follows.

Definition 3.1 (ASP Language of logic programs, $\mathcal{L}_{\mathsf{ASP}}$)**.** *In the following* $\mathcal{L}_{ASP}$ *is a language of* propositional logic *with* propositional symbols*:* $a_0, a_1, \ldots$*;* connectives*: "," (*conjunction*) and meta-connective ";";* disjunction*, denoted as "|";* $\leftarrow$ *(*derivation*, also denoted as* $\rightarrow$*);* propositional constants $\bot$ *(*falsum*);* $\top$ *(*verum*); "*$\neg$*" (*default negation *or* weak negation*, also denoted with the word* not*); "*$\sim$*" (*strong negation*, equally denoted as "*$-$*"*[1]*);* auxiliary symbols*: "(", ")" (*parentheses*). The propositional symbols are called* atoms *too or* atomic propositions*. A* literal *is an atom or a* strong-negated atom*. A* rule *is an ordered pair* $\mathit{Head}(\rho) \leftarrow \mathit{Body}(\rho)$.

An intuitive meaning of *strong negation* "$\sim$" —also called *explicit negation*, *constructible falsity* and *classical negation*— in logic programs with respect to the default negation "$\neg$" is the following: a rule

$$\rho_0 \leftarrow \neg\rho_1$$

allows to derive ρ_0 when there is no *evidence* of ρ_1, while a rule like $\rho_0 \leftarrow \sim\rho_1$ derives ρ_0 only when there is an *evidence* for $\sim\rho_1$, i.e. when it can be proved that ρ_1 is false.

With the notation introduced in Definition 3.1, one may construct clauses of the following general form that are well known in the literature.

Definition 3.2 (Extended Disjunctive Logic Program, EDLP)**.** *An* extended disjunctive logic program *is a set of rules of form*

$$\ell_1 \vee \ell_2 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m, \neg\ell_{m+1}, \ldots, \neg\ell_n \tag{3.1}$$

where ℓ_i *is a literal and* $0 \leq l \leq m \leq n$.

[1] Some times "$-$" is a strong-negation symbol in some *ASP solvers*.

Naturally, an *extended logic program* (or ELP hereafter) is a finite set of rules of form (3.1) with $l = 1$; while an *integrity constraint* (also known in the literature as *strong constraint*) is a rule of form (3.1) with $l = 0$; while a *fact* is a rule of the same form with $l = m = n$. In particular, for a literal ℓ, the *complementary literal* is $\sim\ell$ and vice versa; for a set $\mathcal{M}$ of literals, $\sim\mathcal{M} = \{\sim\ell \mid \ell \in \mathcal{M}\}$, and $Lit_{\mathcal{M}}$ denotes the set $\mathcal{M} \cup \sim\mathcal{M}$; finally, a *signature* $\mathfrak{L}_\Pi$ is a finite set of literals occurring in Π. Additionally, given a set of literals $\mathcal{M} \subseteq \mathcal{A}$, the complement set $\overline{\mathcal{M}} = \mathcal{A} \setminus \mathcal{M}$.

The well-known *semantics of an EDLP* consists of reducing general rules to rules without default negation "$\neg$" because the latter can be interpreted in *classical logic* by means of the well-known *Herbrand models*. In particular, the reduced rules with no default negation Mon of a rule of the form (3.1) is

$$\ell_1 \vee p_1 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m \tag{3.2}$$

where ℓ_i are literals and $0 \leq l \leq m$. This kind of rules is known in the literature as *monotonic counterpart* or *positive program*. Additionally, the monotonic counterpart of a set of rules is the set of the monotonic counterparts of its rules.

Intuitively, the monotonic counterpart is where ASP may coincide with *classical propositional logic*. On the other hand, *default negation* is precisely the main difference between the two systems —alternatively with `Prolog` too. As a result, the corresponding derivation symbols, "$\leftarrow$" for ASP and "$\supset$" for Classical Logic, cannot have the same meaning, as formally stated in the following *orientation principle* —introduced by Brewka and Dix, who identify it as a weak form of *negation-by-failure*:

> If a ground atom α does not unify with some head of a rule from a given program Π, then α is considered to be false. In such a case, $\neg\alpha$ is derivable from Π to distinguish it from classical $\sim\alpha$.

Now let us introduce the meaning of programs with both monotonic and nonmonotonic counterparts.

Suppose a finite ground program Π, consisting of clauses of form (3.1). For any set $\mathcal{S} \subseteq \mathfrak{L}_\Pi$, the *answer-sets reduct* $\Pi^{\mathcal{S}}$ corresponds to

$$\Pi^{\mathcal{S}} = \{\ell_1 \vee \ell_2 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m \mid \{\ell_{m+1}, \ldots, \ell_n\} \cap \mathcal{S} = \emptyset\} \tag{3.3}$$

Stating $\mathcal{S}$ as a set of literals rather than atoms, makes one of the differences with *Stable-models* semantics.

Next, the meaning of a monotonic counterpart corresponds to its *minimal classical model* as follows.

Definition 3.3 (Minimal Closure, Cn(Π))**.** *Let* Π *be a* positive extended disjunctive program *and* $\mathfrak{L}_\Pi$ *the* signature *(set of all ground literals) from* Π*. The set* Cn(Π) *denotes the minimal subset of* $\mathfrak{L}_\Pi$ *where*

1. *for each ground clause* $p_0 \vee p_1 \vee \cdots \vee p_l \leftarrow q_1, \ldots, q_m$ *in* Π, $q_1, \ldots, q_n \in \mathcal{S}$ *implies* $p_i \in \mathcal{S}$ *for some* $0 \leq i \leq l$*; and for each ground clause of the form*

$$\bot \leftarrow q_1, \ldots, q_m \tag{3.4}$$

$\{q_1, \ldots, q_m\} \nsubseteq \mathcal{S}$.

2. *if* $\mathcal{S}$ *contains a pair of complementary literals, then* $\mathcal{S} = \mathfrak{L}_\Pi$.

Note that item (2.) in this Definition 3.3 extends *Stable Models* by giving a meaning to *strong negation*.

Finally, an *answer set* of a given program Π is a *minimal closure* of its reduct as following stated.

Definition 3.4 (Answer Set)**.** *Suppose* Π *is a EDLP and* $\mathcal{S}$ *a set of literals. Then,* $\mathcal{S}$ *is an* answer set *of* Π *if and only if* $\mathcal{S} = Cn(\Pi^{\mathcal{S}})$.

Notice that all stable models can be viewed as *minimal Herbrand models* of a set of first-order sentences, but not the converse. Additionally, $\mathcal{S}$ is a *consistent answer set* of a given program Π if it does not contain a complementary pair of literals.

Last, one may not conclude this section without mentioning that two major *advantages of ASP* over other approaches is a hard work in research both its *declarative programming framework*, and at least two efficient competitive *proving solvers* with long backgrounds: `SMODELS` [Niemela and Simons, 1997] and `DLV` [Calimeri et al., 2002; Leone et al., 2006]. Their respective online running versions may be located at `http://www2.in.tu-clausthal.de/~guadarrama/updates/dlv.html` and at `http://www2.in.tu-clausthal.de/~guadarrama/updates/smodels.html`, which are just (graphical) web interface to the original implemented engines mentioned above. That means they can run on their server and thus there is no need to download and locally install the binaries or sources.

This remarkable asset of *implemented solvers* has come along to study the same problems presented in this work, that makes possible having a formal framework with practical *experimental implementation of updates*, like the one in[Crescini and Zhang, 2005], available at `http://www.cit.uws.edu.au/~jcrescin/projects/PolicyUpdater/`.

3.3 Equivalence in Logic Programming

Checking equivalence between logic programs is of great value, especially when it comes to simplifying them by ignoring some portions of code that might be *redundant* and time-consuming. Moreover, there is a close relation between a particular kind of *equivalence and updates of logic programs*, as discussed along this section.

There are several kinds of equivalence in the literature, particularly in ASP and *monotonic logics* [Eiter et al., 2005; Inoue and Sakama, 2004; Lifschitz et al., 2001;

Osorio et al., 2001]. Since ASP programs may be expressed in some monotonic logics, one may take advantage of checking equivalence in either system. In this dissertation I use $\mathcal{N}_2$-logic as one of its fundamental basis that characterises ASP, as well as a translation function between programs and $\mathcal{N}_2$ theories. The set of axioms may be found in Section 2.1.3

When establishing a relation between $\mathcal{N}_2$ and ASP, a translation function between ASP programs and $\mathcal{N}_2$ theories is necessary. The function is similar to the one from Lifschitz et al.:

Definition 3.5 (Translation into Nelson's logic [Lifschitz et al., 2001])**.** *The mapping function $T_{\mathcal{N}_2}(\cdot)$ translates an EDLP into propositional formulas of Nelson's logic $\mathcal{N}_2$.*

The rule

$$p_0 \vee p_1 \vee \cdots \vee p_l \leftarrow q_1, \ldots, q_m, \neg q_{m+1}, \ldots, \neg q_n$$

is mapped into the formula

$$(q_1 \wedge \cdots \wedge q_m \wedge \neg q_{m+1} \wedge \cdots \wedge \neg q_n) \quad \supset \quad p_0 \vee p_1 \vee \cdots \vee p_l$$

and the strong-negation *propositional symbol "$\sim$" has the same meaning of the logical symbol "$\sim$" in $\mathcal{N}_2$.*

With this translation, one may redefine ASP in terms of $\mathcal{N}_2$-logic, which shall be useful to provide even more features, discussed along this section.

Chapter 2 gives a general picture on how the results of such a characterisation arose: From the logics side, Nelson defined $\mathcal{N}_2$ by introducing *constructible falsity* into *intuitionistic logic*, $\mathcal{H}_{\mathsf{I}}$, and defined $\mathcal{N}_2$ as a least constructive extension to *Here-and-There Logic*, $\mathcal{H}_{\mathsf{HT}}$, which can be defined by extending $\mathcal{H}_{\mathsf{I}}$. Pearce calls it *conservative extension* of $\mathcal{H}_{\mathsf{I}}$ —see Chapter 2 for further details.

From the ASP side, Pearce discovered a new way to define *stable models* in terms of both *intuitionistic logic* and in the logic of *Here-and-There* in [Pearce, 1999a]. Accordingly, as ASP without strong negation is SM, the former can have the same characterisations of SM and vice versa[1]. Besides these characterisations, Pearce discovered that by introducing *strong negation*, like in Definition 3.3, Nelson's $\mathcal{N}_2$-logic characterises ASP as well.

The main result of such characterisations that is more relevant to this thesis, can be expressed by Theorem 3.1, by using the notation from Osorio and Cuevas, and originally due to Pearce, introduced also in Section 2.1.3.

To begin with, the notation $\tau \Vdash_{\mathcal{N}_2} \mathcal{M}$ is a shorthand for both τ is *consistent* and *derives* $\mathcal{M}$ in $\mathcal{N}_2$-logic.

[1]Note that *strong negation* can be introduced in SM very easily, with extra atoms and *integrity constraints*.

Theorem 3.1 ([Osorio and Cuevas, 2007]). *Let* Π *be a program over a set of atoms* $\mathcal{A}$ *and* $\mathcal{M} \subseteq \mathfrak{L}_{\mathcal{A}}$ *a consistent set of literals. The set* $\mathcal{M}$ *is an answer set of* Π *if and only if* $T_{\mathcal{N}_2}(\Pi) \cup \neg\overline{\mathcal{M}} \cup \neg\neg\mathcal{M} \Vdash_{\mathcal{N}_2} \mathcal{M}$.

With such a theorem, one may easily establish an equivalence relation between ASP programs for some upcoming update properties:

Theorem 3.2 ([Lifschitz et al., 2001]). *For any programs* Π_1 *and* Π_2, $T_{\mathcal{N}_2}(\Pi_1) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_2)$ *if and only if for every program* Π, $\Pi \cup \Pi_1$ *and* $\Pi \cup \Pi_2$ *have the same Answer Sets.*

In order to simplify notation and with a slight abuse of notation, for any ASP programs Π_0, Π_1, $\Pi_0 \equiv_{\mathcal{N}_2} \Pi_1$ shall actually stand for $T_{\mathcal{N}_2}(\Pi_0) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_1)$.

This theorem expresses how two programs may have a *logical content* in $\mathcal{N}_2$ logic, whose axioms are introduced in Section 2.1.3. For practical reasons, however, one might want to avoid introducing more axioms and work just at a simpler side of ASP. That is to say, the following paraphrase of Theorem 3.2 is also possible:

Theorem 3.3 (Strong Equivalence [Lifschitz et al., 2001]). *For any programs* Π_1 *and* Π_2, Π_1 *is strongly equivalent to* Π_2 *if and only if for every program* Π, $\Pi \cup \Pi_1$ *and* $\Pi \cup \Pi_2$ *have the same Answer Sets.*

Now one can define at least *three kinds of equivalence* that are useful in the properties of updates semantics presented in this dissertation. The most basic one in this set of three, $\Pi_1 \equiv_{\mathsf{ASP}} \Pi_2$, means that Π_1 and Π_2 have the same answer sets, and is known in the literature as *weak equivalence* of logic programs, also denoted as simply "$\equiv$". Secondly, $\Pi_1 \equiv_{\mathcal{N}_2} \Pi_2$ means that programs Π_1 and Π_2 have *identical Answer Sets* for any set of rules added to them. This corresponds to *strong equivalence* in the literature. Finally, a third kind of equivalence has been introduced by Inoue and Sakama that, besides characterising addition of logic programs, it may also capture removal operations. They named it *strongly-update equivalence* and it is formalised in my own notation as follows.

Definition 3.6 (S-strong equivalence [Inoue and Sakama, 2004]). *Programs* Π_1 *and* Π_2 *are* strongly-update equivalent *(or S-update equivalent) if for any programs* Π_x *and* Π_y, $(\Pi_1 \setminus \Pi_x) \cup \Pi_y$ *and* $(\Pi_2 \setminus \Pi_x) \cup \Pi_y$ *have the same answer sets.*

The authors give an example where a strongly-equivalent logic program cannot be S-update equivalent to another: $\{p \vee p \leftarrow q\}$ is strongly-equivalent to $\{p \leftarrow q\}$. However, it is easy to test that these two programs are not S-update equivalent.

By following Definition 3.6, Inoue and Sakama noticed that two S-update equivalent programs, Π_1 and Π_2 are strongly equivalent when $\Pi_x = \emptyset$. In addition, they are weakly equivalent if and only if $\Pi_x = \Pi_y = \emptyset$.

Note that $\equiv_{\mathcal{N}_2}$ is much stronger than $\equiv_{\mathsf{ASP}}$, and replacing $T_{\mathcal{N}_2}(\Pi_1) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_2)$ with $\Pi_1 \equiv_{\mathsf{ASP}} \Pi_2$ *is not correct.* Take for example, $\Pi = \{b \leftarrow \top\}$, $\Pi_1 = \{(a \leftarrow$

$\top); (b \leftarrow b)\}$, $\Pi_2 = \{(a \leftarrow \neg b);\ (b \leftarrow b)\}$. Clearly, $\Pi_1 \equiv_{\mathsf{ASP}} \Pi_2$ holds. However, an *arbitrary semantics* $Sem(\cdot)$ for updating Π with Π_1 should correspond to $Sem(\Pi \diamond \Pi_1) = \{a, b\}$, and updating Π with Π_2 should correspond to $Sem(\Pi \diamond \Pi_2) = \{\neg a, b\}$. As a result, $Sem(\Pi \diamond \Pi_1) \equiv_{\mathsf{ASP}} Sem(\Pi \diamond \Pi_2)$ does *not* hold! Therefore, a *weak equivalence* between updating logic programs is not enough to produce the same resulting update. A stronger notion of equivalence between the two updates is necessary, simply because belief changes (to nonmonotonic theories imply a dynamics.

In summary, this section is an introduction of several kinds of *equivalence of logic programs*, its importance and relevance in belief changes. These different settings of equivalence may help simplify logic programs when some portions of code are redundant. There are also helpful equivalence-settings that can capture dynamic changes, and thus relevant to the study of this dissertation. Finally, a logic characterisation of ASP has been considered in this section as an alternative tool to *equivalence-checking of logic programs* by translating them into $\mathcal{N}_2$-propositional theories.

3.4 Weak Constraints

Leone et al. introduced a nice feature of `DLV` solver [see Buccafurri et al., 2000; Leone et al., 2006] known as *Weak Constraints* that may be employed to set up preferences between models. In particular, a *weak constraint* is a variant of an *integrity constraint* that may be violated in order to establish priorities amongst models. One of its differences is the introduction of a new derivation symbol "$:\sim$", rather than "$:-$ " or "$\leftarrow$". Moreover, one can specify the priority level and weight of the constraint. Formally,

Definition 3.7 (Weak Constraint [Leone et al., 2006])**.** *A* weak constraint *(ω) is an expression of the form*

$$:\sim \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m[w : p] \tag{3.5}$$

where for $0 \leq k \leq m$, $\ell_1, \ldots, \ell_m$ are literals, while w (the weight) and p (the level, or layer) are positive integer constants or variables. For convenience, w and/or p may be omitted and are set to 1 in such a case.

In addition, $\Omega(\Pi)$ shall denote the finite *set of weak constraints* occurring in a given program Π. Likewise, a *ω-program* is a logic program with weak constraints.

The intuition behind interpretations of ω-programs consists in minimising the sum of weights of violated weak constraints at the highest *priority level*, and amongst them those which minimise the sum of weights of the violated weak constraints in the next lower level, and so on.

Next, it is worth noticing the introduction of a new symbol "$:\sim$" to specify a *weak derivation*, very different from the classical `Prolog` *derivation symbol*: "$:-$ ". Both of them, however, are widely used in ASP solvers.

In order to provide a more syntactic sugar, another way to define a weak-consitraint expression from Definition 8.10 is as follows.

Definition 3.8 (Weak Constraint)**.** *A* weak constraint *(ω) is an expression of the form*

$$[w : p] \leftarrow \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m \tag{3.6}$$

where for $0 \leq k \leq m$, $\ell_1, \ldots, \ell_m$ are literals, while w (the weight) and p (the level, or layer) are positive integer constants or variables. For convenience, when w and/or p are omitted, they are set to 1*.*

From now on, the previous weak-constraints form shall be employed in the context of `DLV`-code, while the other in higher abstraction levels.

Similarly to integrity constraints in Section 3.1, one may say that a weak constraint $\rho = ([w : p] \leftarrow \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m)$ is *violated* by an answer set $\mathcal{S}$ of a program Π if the following three conditions hold:

1. $\rho \in \Pi$
2. $\{\ell_1, \ldots, \ell_k\} \subseteq \mathcal{S}$
3. $\{\ell_{k+1}, \ldots, \ell_m\} \nsubseteq \mathcal{S}$

Additionally, Leone et al. simplify the combination of weights in levels by introducing a function $H^{\Pi}(\mathcal{S})$ that grows in direct proportion to the weight and level of the weak constraint as follows:

Definition 3.9 (Objective Function, $H^{\Pi}(\mathcal{S})$ [Leone et al., 2006])**.** *Given a ground program Π with weak constraints $\Omega(\Pi)$ and an answer set $\mathcal{S}$, the ω* objective function *$H^{\Pi}(\mathcal{S})$ for Π is a product sum and is defined by using an auxiliary function f_{Π} that maps levelled weights to weights without levels:*

$$\begin{aligned} f_{\Pi}(1) &= 1 \\ f_{\Pi}(n) &= f_{\Pi}(n-1) \cdot |\Omega(\Pi)| \cdot w^{\Pi}_{max} + 1, n > 1 \\ H^{\Pi}(\mathcal{S}) &= \sum_{i=1}^{l^{\Pi}_{max}} (f_{\Pi}(i) \cdot \sum_{\rho \in N^{\Pi}_i(\mathcal{S})} weight(\rho)) \end{aligned}$$

where w^{Π}_{max} and l^{Π}_{max} denote the maximum weight and maximum level over the weak constraints in Π, respectively; $N^{\Pi}_i(\mathcal{S})$ denotes the weak constraints at level i violated by $\mathcal{S}$, and $weight(\rho)$ the weight of weak constraint ρ.

Leone et al. states that $|\Omega(\Pi)| \cdot w^{\Pi}_{\max} + 1$ is greater than the sum of all weights in the program, and therefore guaranteed to be greater than the sum of weights of any single level.

Finally, the *best models* of such a logic program are those that *minimise* the number of *violated weak constraints*. The definition of a *weak-constraint model* from Buccafurri et al. in my own notation is the following.

Definition 3.10 (Weak-Constraint Model [Leone et al., 2006])**.** *For an EDLP* Π *with weak constraints, a set* $\mathcal{S}$ *is a* weak-constraint model *of* Π *if and only if*

1. $\mathcal{S}$ *is an answer set of* Π
2. $H^{\Pi}(\mathcal{S})$ *is minimal over all the answer sets of* Π.

When the underlying semantics is ASP in Definition 8.13, a weak-constraint model is also known as *Optimal Answer Set*.

Moreover, the language of EDLP's with weak constraints shall be called $\mathsf{DATALOG}^{\vee,\omega}$, which is very similar to the notation from the literature.

3.5 Ordered Disjunctions

Ordered-disjunctive Logic Programming, ODLP by [Brewka, 2002; Brewka et al., 2004], is an extension to ELP's and may be defined in broad way as follows: a simple *ordered-disjunction program* is a set of rules of form

$$C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_k$$

where C_i, A_j and B_l are all ground literals. $C_1, \ldots, C_n$ are usually named the *choices of a rule* and their intuitive reading is as follows: The ordered disjunction is used only in rule heads to select some of the answer sets of a program as the preferred ones. If C_1 is possible, then C_1; if C_1 is not possible, then try C_2; ...; if neither $C_i, \ldots, C_{n-1}$ is possible then try C_n. Moreover, one may identify some special cases such as: if $n = 0$ the rule is a *constraint*; and finally, facts are those rules where $m = k = 0$. In the particular case of this dissertation, the required codification of ordered disjunctive programs shall be just $n = 2, m = k = 0$, as in Section 6.3.5.

3.5.1 ODLP-reduct

In particular, there is a reduct of ODLP-rules with respect to a set of literals, as well as a reduct of ODLP-programs:

Definition 3.11 ($\times$-reducts, [Brewka et al., 2004])**.** *Let*

$$\rho = C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_k$$

be a rule and $\mathcal{M}$ *a set of literals. Then, the* $\times$*-reduct* $\rho_{\times}^{\mathcal{M}}$ *of* ρ *is defined as*

$$\rho_{\times}^{\mathcal{M}} = \{C_i \leftarrow A_1, \ldots, A_m \mid C_i \in \mathcal{M} \text{ and } \mathcal{M} \cap \{C_1, \ldots, C_{i-1}, B_1, \ldots, B_k\} = \emptyset\}.$$

Let Π *be an ODLP. The* $\times$*-reduct* $\Pi_{\times}^{\mathcal{M}}$ *of* Π *is defined as:*

$$\Pi_{\times}^{\mathcal{M}} = \bigcup_{\rho \in \Pi} \rho_{\times}^{\mathcal{M}}.$$

One may compute the *answer sets of a* $\times$*-reduct,* $\Pi_{\times}^{\mathcal{M}}$ as possible interpretations of an ODLP, by means of the following proposition.

Proposition 3.1 ([Brewka et al., 2004])**.** *Let* Π *be an ODLP and* $\mathcal{M}$ *a set of literals. Then,* $\mathcal{M}$ *is an answer set of* Π *if and only if the following three conditions hold:*

1. $\mathcal{M} = Cn(\Pi_{\times}^{\mathcal{M}})$,
2. $\mathcal{M}$ *is consistent, and*
3. $\mathcal{M}$ *satisfies every rule* $\rho \in \Pi$.

Finally, the models of an ODLP are defined in terms of *preferred answer sets* in at least three categories with respect to its *satisfaction degree*. Namely, *cardinality-preference*, *inclusion-preference* and *paretto-preference*. In this dissertation, the focus is on the first two of them, as later explained in Section 6.3.5 and Section 8.4.1.

3.5.2 ODLP-semantics

Before introducing semantics to characterise ODLP's, it is necessary to define what a *satisfaction degree* is.

Definition 3.12 (Degree of satisfaction, $\mathsf{d}_{\mathcal{M}}(\rho)$ [Brewka et al., 2004])**.** *Let* $\mathcal{M}$ *be an answer set of an ODLP. The satisfaction degree* $\mathsf{d}_{\mathcal{M}}(\rho)$ *by* $\mathcal{M}$ *of a rule* ρ *of form*

$$C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_k$$

- *is* 1 *if* $A_j \notin \mathcal{M}$*, for some* j*, or* $B_i \in \mathcal{M}$ *for some* i*;*
- *is* j *with* $1 \leq j \leq n$*, if all* $A_j \in \mathcal{M}$*, no* $B_i \in \mathcal{M}$*, and*

$$j = \min\{r \mid C_r \in \mathcal{M}\}$$

With the degree of satisfaction, one can define many *preference relations*, as the ones suggested by Brewka et al.. For that purpose, they also define the set of rules with a degree of satisfaction as follows: Given a set of literals $\mathcal{M}$, let $\mathcal{M}^i(\Pi) = \{\rho \in \Pi \mid \mathsf{d}_{\mathcal{M}}(\rho) = i\}$.

Definition 3.13 (Cardinality Preference [Brewka et al., 2004]). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be answer sets of an ODLP, Π. Then, $\mathcal{M}_1$ is cardinality-preferred to $\mathcal{M}_2$ (denoted as $\mathcal{M}_1 >_c \mathcal{M}_2$) if and only if there is an i such that $|\mathcal{M}^i(\Pi)_1| >_c |\mathcal{M}^i(\Pi)_2|$; and for all $j < i$ and $|\mathcal{M}^j(\Pi)_1| = |\mathcal{M}^j(\Pi)_2|$.*

In other words, model $\mathcal{M}_1$ is *cardinality-preferred* to $\mathcal{M}_2$ from program Π if and only if there is a *satisfaction degree* for which a set of rules satisfied by $\mathcal{M}_1$ is bigger than the set of rules satisfied by $\mathcal{M}_2$. The inclusion preference has a similar intuition:

Definition 3.14 (Inclusion Preference [Brewka et al., 2004]). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be answer sets of an ODLP, Π. Then, $\mathcal{M}_1$ is inclusion-preferred to $\mathcal{M}_2$ ($\mathcal{M}_1 >_i \mathcal{M}_2$) if and only if there is an i such that $\mathcal{M}^i(\Pi)_2 \subset \mathcal{M}^i(\Pi)_1$; and for all $j < i$ and $\mathcal{M}^j(\Pi)_1 = \mathcal{M}^j(\Pi)_2$.*

Brewka et al. also state the following important relation.

Proposition 3.2 (Brewka et al.). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be answer sets of an ODLP, Π. Then $\mathcal{M}_1 >_i \mathcal{M}_2$ implies that $\mathcal{M}_1 >_c \mathcal{M}_2$.*

Finally, the indented models of an ODLP are called *k-preferred answer sets* and are based on the satisfaction degree and the preference criterion. Formally,

Definition 3.15 (Preferred Answer Sets, Brewka et al.). *A set of literals $\mathcal{M}$ is a k-*preferred answer set *(where $k \in \{c, i\}$) of an ODLP, Π, if and only if $\mathcal{M}$ is an answer set of Π and there is no answer set $\mathcal{M}'$ of Π such that $\mathcal{M}' >_k \mathcal{M}$.*

There are more preference criterions, but the two definitions just introduced are the most relevant to this dissertation.

3.5.3 ODLP and Weak Constraints

There is a very-interesting alternative to preferred answer sets by means of weak constraints. For this end, Brewka et al. have found a relation between them. They propose a translation of a normal logic program Π with weak constraints into an ODLP, Π', by replacing every weak constraint ω of form (3.6) —with $w = p = 1$— into the following pair of rules:

$$\alpha_\omega \times {\sim}\alpha_\omega \leftarrow \top \tag{3.7}$$

$$\bot \leftarrow \alpha_\omega, b_1, \ldots, b_k, \neg b_{k+1}, \ldots \neg b_m \tag{3.8}$$

where α_ω is a new atom that denotes the constraint ω is not violated.

Accordingly, Brewka et al. state that *ω-preference* corresponds to *c-preference* because the amount of violated weak constraints is exactly the same amount of rules satisfied to the second degree [Brewka et al., 2004]. Such correspondence may be useful for the implementations of both Section 6.3.5 and Section 8.4.2, in upcoming chapters.

3.5.4 ODLP-solver

Finally, as one of the distinguishing features of logic programming, it is very important to notice that `PSmodels`[1] is an *ODLP implemented prototype* [Brewka et al., 2002], which consists of an extension to `SMODELS`[2] [Niemela and Simons, 1997] to compute *preferred stable models* of normal logic programs. However, it is also important to point out that the sources themselves need some *debugging*[3].

PSmodels, in a bug-free implementation however, should compute *preferred stable models* (also known as *k-preferred answer sets*) of normal logic programs under ODLP and should be considered when thinking of implementing the update semantics proposed in this dissertation.

One of the features of `PSmodels` is to tell how many times the test program has been invoked to check whether a stable model of a given ODLP is a preferred one. Besides the original implementation sources available at the author's sites, there is a compiled *online front-end* running at `http://www.in.tu-clausthal.de/~guadarrama/updates/psmodels.html` that provides the original compiled `PSmodels` to run as an *Internet service* in a webpage, rather than the classical online run. This overcomes the need to download it, compile it and run it locally.

3.6 Abductive Programming and GAS

This section is a recapitulation of some basic definitions about syntax and semantics of abductive logic programs. *Abduction* is an alternative framework to *deductive reasoning* in Classical Logic [Kakas and Mancarella, 1990] whose general intuition consists in forming *hypotheses* and choosing the best ones that explain *observations* or conclusions. This intuition will prove to be very useful to generalise the simple formulation for updates of logic programs here presented.

As one of the semantics to interpret abductive programs, *Minimal Generalised Answer Sets* (MGAS) provides a more general and flexible semantics than standard ASP, with a wide range of applications. In particular, this framework has been employed to restore consistency [Balduccini and Gelfond, 2003], to set up preferences [Osorio et al.,

[1]The sources may be downloaded from `http://www.tcs.hut.fi/Software/smodels/priority` and there is a graphical user front end at `http://www.in.tu-clausthal.de/~guadarrama/updates/psmodels.html` that allows to execute preferred logic programs online.

[2]This solver may be downloaded from `http://www.tcs.hut.fi/Software/smodels/` and run via online with a graphical user front end at `http://www.in.tu-clausthal.de/~guadarrama/updates/smodels.html`.

[3]For instance, Version 2.26a will crash with a simple program like $\{a.\}$.

2004], as well as in an earlier proposal to perform (limited) updates [Guadarrama et al., 2005], for instance.

Definition 3.16 (Abductive Logic Program [Kakas and Mancarella, 1990])**.** *An* abductive logic program *is a pair* $\langle\, \Pi, \mathcal{A}^* \,\rangle$ *where* Π *is an arbitrary program and* $\mathcal{A}^*$ *a set of literals, called abducibles.*

As a consequence, one may extend the well-known standard *signature* definition with $\mathfrak{L}_{\langle\, \Pi, \mathcal{A}^* \,\rangle}$, which means the finite set of literals occurring both in Π and in $\mathcal{A}^*$.

On the other hand, there already exists a semantics to interpret abductive programs, called *generalised answer sets* (GAS) due to Kakas and Mancarella. The intuition behind it consists of merging combinations of abducibles with an original program. Then, the resulting program(s) may be interpreted in ASP, as formally expressed in the following definition.

Definition 3.17 (GAS [Kakas and Mancarella, 1990])**.** *The expression* $\mathcal{M}(\Delta)$ *is a* generalised answer set *of the abductive program* $\langle\, \Pi, \mathcal{A}^* \,\rangle$ *if and only if* $\Delta \subseteq \mathcal{A}^*$ *and* $\mathcal{M}(\Delta)$ *is an answer set of* $\Pi \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$.

In case of more than one generalised answer sets, a *preferred inclusion order* may be established.

Definition 3.18 (Abductive Inclusion Order [Balduccini and Gelfond, 2003])**.** *Let* $\mathcal{M}(\Delta_1)$ *and* $\mathcal{M}(\Delta_2)$ *be generalised answer sets of* $\langle \Pi, \mathcal{A}^* \rangle$*. The relation* $\mathcal{M}(\Delta_1) \leq_{\mathcal{A}^*} \mathcal{M}(\Delta_2)$ *holds if and only if* $\Delta_1 \subseteq \Delta_2$.

From now on, let us introduce a "sweeter" syntactic sugar to represent *generalised answer sets* by $\mathcal{M}_\Delta$, where $\mathcal{M}$ is the resulting answer set with the abducible set Δ. This is equivalent to the well known notation $\mathcal{M}(\Delta)$, but in a form that uses less paper.

Example 3.1. *Let* $\{a, b\}$ *be abducibles and* Π *be* $\{(a \leftarrow b), (b \leftarrow a), (c \leftarrow a)\}$*. Then* $\{a, b, c\}_{\{a\}}$ *—that is, the resulting answer set* $\{a, b, c\}$ *with* $\{a\}$ *as abducible— is a* GAS *of* $\langle \Pi, \{a, b\} \rangle$*, since* $\{a, b, c\}$ *is an answer set of* $\Pi \cup \{a \leftarrow \top\}$*, as well as* $\{a, b, c\}_{\{a,b\}}$ *and* $\{\}_{\{\}}$*. Therefore,* $\{a, b, c\}_{\{a\}} \leq_{\mathcal{A}^*} \{a, b, c\}_{\{a,b\}}$*, since* $\{a\} \subseteq \{a, b\}$*. However,* $\{\}_{\{\}}$ *is the minimal* GAS *of* Π*, as* $\emptyset$ *is a subset of any set. Finally, note that* $\{a, b, c\}_{\{c\}} \not\leq_{\mathcal{A}^*} \{a, b, c\}_{\{a,b\}}$

Last, one can easily establish the *minimal generalised answer sets* from an abductive inclusion order with the following definition

Definition 3.19 (MGAS [Balduccini and Gelfond, 2003])**.** *Let* $\mathcal{M}(\Delta)$ *be a* minimal generalised answer set *(MGAS) of* $\langle\, \Pi, \mathcal{A}^* \,\rangle$ *if and only if* $\mathcal{M}(\Delta)$ *is a generalised answer set of* $\langle\, \Pi, \mathcal{A}^* \,\rangle$ *and it is minimal with respect to its* abductive inclusion order.

It is worth mentioning that minimal generalised answer sets define the semantics of `CR-Prolog`, where Balduccini and Gelfond characterised *Consistency Restoring Rules* by means of MGAS.

This simple and strong framework is the main core of a solid foundation for the update formulation, presented in the following sections.

3.7 Complexity Notation

The following two sections are a recapitulation of Johnson's complexity hierarchy. Further concepts can also be found in [Cai, 2003].

3.7.1 The Polynomial Hierarchy

To begin with, the classes of complexity relevant to this thesis can be briefly summarised as follows. There is a class with name P that denotes the set of problems from which a solution can be found in *polynomial time* by a *deterministic Turing machine*, also denoted as PSPACE and PTIME, depending on which of the two most common resources (time and space) is being considering. Next, NP denotes the class of problems solved in polynomial time by a *nondeterministic Turing machine*, which usually correspond to *guess/decision problems*, also denoted as NPSPACE and NPTIME, again depending on which of the two most common resources is being considered. Accordingly, the class coNP denotes the set of decision problems whose complement is in NP-time. In the literature they say that x is an NP-hard problem if for some NP-complete problem there is a polynomial-time Turing reduction from it to x, which usually corresponds to *search problems*. In addition, an NP-hard problem is defined likewise.

On the other hand, under the assumption that $\mathrm{P} \neq \mathrm{NP}$ and that $\mathrm{P} \neq \mathrm{coNP}$, one can establish that $\mathrm{P} \subseteq \mathrm{NP}$ and $\mathrm{P} \subseteq \mathrm{coNP}$. As a result, a problem is said to be in P-*tractable* or in NP-hard vs. coNP-hard *intractable* for the case of being solved in a *super-polynomial time*.

This is the basis of further superclasses. In particular, one in which the problems of this thesis reside is known in the literature as *polynomial hierarchy*. According to Johnson, they are due to Meyer and Stockmeyer and denoted as follows:

$$\Delta_0^{\mathrm{P}} = \Sigma_0^{\mathrm{P}} = \Pi_0^{\mathrm{P}} = \mathrm{P}$$

and for $0 \leq k$,

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

where NP^C corresponds to the class of decision problems that can be solved in a polynomial time by a nondeterministic Turing machine using a cost-free (one-step) *oracle* (*strategy*) in the class C; while Π_{k+1}^{P} corresponds to the complement of the class of problems that can be solved in $\Sigma_{k+1}^{\mathrm{P}}$ Accordingly, a problem x is complete in class C if $x \in C$ and for every problem $y \in C$, it has a polynomial reduction to x. Finally, it is

easy to see that

$$\Delta_1^{\mathrm{P}} = \mathrm{P};\ \Sigma_1^{\mathrm{P}} = \mathrm{NP};\ \Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}};\ \Pi_1^{\mathrm{P}} = \mathrm{coNP};\ \Delta_2^{\mathrm{P}} = \mathrm{P}^{\mathrm{NP}}.$$

In general, Johnson defines the class $\mathbf{PH} = \bigcup_{k=0}^{\infty} \Sigma_k^{\mathrm{P}}$ that makes sense with the earlier supposition $\mathrm{P} \neq \mathrm{NP}$. Otherwise, if $\mathrm{P} = \mathrm{NP}$, then $\Sigma_{k+1}^{\mathrm{P}} = \Sigma_k^{\mathrm{P}} = \mathrm{P}$ with $0 \leq k$, and consequently, it is said that the hierarchy *collapses* to P and $\mathrm{P} = \mathbf{PH}$. Back to the supposition that $\mathrm{P} \neq \mathrm{NP}$, the hardest problems in **PH** are said to be *intractable*, as previously mentioned.

On the other hand, for all $1 \leq k$,

$$\Sigma_k^{\mathrm{P}} \subseteq \Theta_{k+1}^{\mathrm{P}} \subseteq \Delta_{k+1}^{\mathrm{P}} \subseteq \Sigma_{k+1}^{\mathrm{P}} \subseteq \mathrm{PSPACE},$$

by considering the initial conjectures. Notice that the class Θ_k^{P} (also known as $\Delta_k^{\mathrm{P}}[O(\log n)]$) is a refinement to the classes Δ_k^{P} for $k \geq 2$, where the number of calls to the oracle in each computation is bounded by $O(\log n)$ to the size of the input, n [Buccafurri et al., 2000].

3.7.2 The Exponential-time Hierarchy

This section is a brief introduction to the complexity notation to describe the classes of intractable problems, whose execution time grows exponentially in the size of the input, growing faster than any polynomial time.

According to Johnson, that class can be defined as

$$\mathrm{EXPTIME} = \bigcup_{k>0} \mathrm{TIME}[2^{n^k}].$$

That is to say, EXPTIME is the set of all decision problems whose execution time is bounded by $2^{p(n)}$, where p is a polynomial. Accordingly, the nondeterministic definition is trivial: $\mathrm{NEXPTIME} = \bigcup_{k>0} \mathrm{NTIME}[2^{n^k}]$.

Finally, higher-hierarchy classes continue to grow to 2-EXPTIME, and 2-NEXPTIME respectively, where

$$\text{2-EXPTIME} = \bigcup_{k>0} \mathrm{TIME}[2^{2^{n^k}}]$$

to constitute a 2-EXPSPACE and 2-NEXPSPACE. In general, a called *elementary* class is defined in [Johnson, 1990] as

$$\mathrm{ELEMENTARY} = \bigcup_{k>1} k\text{-EXPTIME}.$$

Remember that his own religion is the truest to every man even if it stands low in the scales of philosophical comparison.

Mohandas Gandhi

Chapter 4

A Road Map for Update Semantics

As one of the major and traditional topics in Artificial Intelligence over the last years, *knowledge representation and reasoning* has proved to be a strong theoretical framework to manage knowledge bases. As a result, this particular topic has become more widely studied in administration of knowledge bases of intelligent (rational) agents, especially in situations of incomplete knowledge from a changing environment, and this area of research is known in the literature as *belief update*.

The history of *semantics for updates* of logic programs is rather long. Indeed, it starts in the days of some of the first versions of `Prolog` with its commands assert and retract. However, sooner they started to yield conflicting information and other (unexpected) *side effects*. It was also time of research on *databases* with publications like [Fagin et al., 1983], and in particular for *logical databases*: [Fagin, 1995; Fagin et al., 1986; Winslett, 1990]. Nevertheless, some of the first formalisms to carry out proper changes to *monotonic theories* have been originally studied by Alchourrón et al.; Katsuno and Mendelzon; Katsuno and Mendelzon; Lehmann; Makinson, while in the *non-monotonic side* by Kraus et al.; Lehmann and Magidor; Makinson.

On the other hand, Gelfond and Lifschitz formulated the *Stable Models Semantics* in [Gelfond and Lifschitz, 1988] (also refereed as *Answer Sets Semantics*, SM or simply *ASP*), and more concrete proposals arose within that framework, aimed at the problem of updating knowledge: [Alferes et al., 1999; Eiter et al., 2000a,b, 2001, 2002; Osorio and Zacarías, 2003; Sakama and Inoue, 1999, 2003; Zhang, 1995, 2001, 2006].

The purpose of this chapter is to introduce current and some of those past proposals to update logic programs (or alike), by pointing out features as well as some of their limitations to represent *correct evolving knowledge*. Nevertheless, this survey is just a small thread of a massive research over more than two decades, and by no means exhaustive. It just takes into account those proposals that are the most relevant and of interest for this thesis.

4.1 Eiter and Others

To the best of my knowledge, Eiter et al. achieved the most complete survey of most known semantics for updating logic programs, by gathering relevant postulates and principles from the literature. The approach first appeared in [Eiter et al., 2000b] with a vast study of well-known and well-accepted postulates and properties, and later refined in [Eiter et al., 2002] and extended to be a main component in more general problems like agents in [Eiter et al., 2005] or preferences in [Eiter et al., 2002], and they also implemented a solver available at `http://www.kr.tuwien.ac.at/staff/giuliana/project.html#Download` that is the main engine of an experimental graphical front end from us at `http://www2.in.tu-clausthal.de/~guadarrama/updates/upd.html`.

One of the main assets of Eiter et al.'s proposal, as already mentioned, is being one of the first or even the first to realise a deep study of the literature of belief change of logic programs, in particular in ASP, most of such study is already introduced in Chapter 2.

Eiter et al. formulate a natural definition for updating logic program sequences on a restricted Answer Sets language by rejecting rules under a *causal rejection principle*. The principle is due to Alferes et al. that later, however, turned out to be *counterintuitive*, even to themselves: [see Alferes et al., 2005; Eiter et al., 2005; Osorio and Zacarías, 2003]. This "counter-intuition" comes from their strong dependency in the syntax of programs, according to Eiter et al. Section 4.2 includes further discussion about this claim.

In particular, the natural formula under which Eiter et al. analyse and describe update properties comes from [Eiter et al., 2002] as follows.

Given an update sequence $(\Pi_1, \Pi_2, \ldots, \Pi_n)$, with $n \leq 2$, over a set of atoms $\mathcal{A}$, assume $\mathcal{A}^*$ as an extension of $\mathcal{A}$ by new pair-wise unique atoms $\mathsf{rej}(\rho)$; α_i, for each rule ρ occurring in Π; each atom $\alpha \in \mathcal{A}$, and $1 \leq i \leq n$. An injective naming function $\mathsf{Name}(\cdot, \cdot)$ is also assumed, which assigns to each rule ρ in a program Π_i a unique name, $\mathsf{Name}(\rho, \Pi_i)$, provided that $\mathsf{Name}(\rho, \Pi_i) \neq \mathsf{Name}(\rho', \Pi_j)$ whenever $i \neq j$. Finally, for a literal ℓ, ℓ_i denotes the result of replacing an atomic formula α of ℓ by α_i.

The intuitive idea of $\mathsf{rej}(\rho)$ is that of an atom that blocks (rejects or *inhibits*) a related rule ρ when true, provided that there is another more recent rule ρ' with *conflicting information*.

Definition 4.1 (Update program [Eiter et al., 2002])**.** *Given an update sequence*

$$(\Pi_1, \Pi_2, \ldots, \Pi_n)$$

over a set of atoms $\mathcal{A}$, the update program $\mathbf{\Pi}_{\lhd} = (\Pi_1 \lhd \cdots \lhd \Pi_n)$ *over $\mathcal{A}^*$ consists of the following items:*

(i) all constraints in Π_i, $1 \leq i \leq n$;

(ii) for each $\rho \in \Pi_i$, $1 \leq i \leq n$*:*

$$\ell_i \leftarrow \mathit{Body}(\rho), \neg \mathit{rej}(\rho) \qquad \text{if } \mathit{Head}(\rho) = \ell;$$

(iii) for each $\rho \in \Pi_i$, $1 \leq i < n$*:*

$$\mathit{rej}(\rho) \leftarrow \mathit{Body}(\rho), \neg \ell_{i+1} \qquad \text{if } \mathit{Head}(\rho) = \ell;$$

(iv) for each literal ℓ *occurring in* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ $(1 \leq i \leq n)$*:*

$$\ell_i \leftarrow \ell_{i+1}; \qquad \ell \leftarrow \ell_i.$$

Note that in *(iv)* the authors write ℓ_1 rather than ℓ_i. Moreover, at the same *(iv)* they write $1 \leq i < n$ instead of $1 \leq i \leq n$. Zhang also detected and corrected these typos in [Zhang, 2006]. Lastly, they do not state how to treat double negations that might happen in *(iii)*.

Next, Eiter et al. define the intended answer sets of an update sequence $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ in terms of the answer sets of $\mathbf{\Pi}_\lhd = (\Pi_1 \lhd \cdots \lhd \Pi_n)$. In other words, the models are back to the original alphabet by filter them out with the original atoms:

Definition 4.2 (Answer sets of an update sequence [Eiter et al., 2002])**.** *Let*

$$(\Pi_1, \Pi_2, \ldots, \Pi_n)$$

be an update sequence over a set of atoms $\mathcal{A}$*. Then,* $\mathcal{S} \subseteq \mathit{Lit}_\mathcal{A}$ *is an* update answer set *of* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *if and only if* $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ *for some answer set* $\mathcal{S}'$ *of* $\mathbf{\Pi}_\lhd = (\Pi_1 \lhd \cdots \lhd \Pi_n)$*. The collection of all of the update answer sets of* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *is denoted by* $\mathcal{U}((\Pi_1, \Pi_2, \ldots, \Pi_n))$*.*

There is a solver available for downloading at `http://www.kr.tuwien.ac.at/staff/giuliana/project.html#Download` and I have installed it to run online at `http://www2.in.tu-clausthal.de/~guadarrama/updates/upd.html`, which also provides a graphic-oriented interface on the server itself. Obviously, no download or installation is necessary to run the latter version.

Supposing the corrected semantics is what the authors wanted, computing their following example is possible:

Observation 4.1. *[Eiter et al., 2002] Assume a daily update regarding an energy flaw*

represented by the sequence (Π_1, Π_2) *where*

$$\begin{aligned}
\Pi_1 = \{sleep &\leftarrow night, \neg tvon\\
night &\leftarrow \top\\
watchtv &\leftarrow tvon\\
tvon &\leftarrow \top\}\\
\Pi_2 = \{\sim tvon &\leftarrow pfailure\\
pfailure &\leftarrow \top\}
\end{aligned}$$

by Definition 4.1, the update program $\mathbf{\Pi}_{\triangleleft} = (\Pi_1 \triangleleft \cdots \triangleleft \Pi_n)$ *consists of the following rules:*

$$\begin{aligned}
sleep_1 &\leftarrow night, \neg tvon, \neg \mathit{rej}(\rho_1)\\
night_1 &\leftarrow \neg \mathit{rej}(\rho_2)\\
watchtv_1 &\leftarrow tvon, \neg \mathit{rej}(\rho_3)\\
tvon_1 &\leftarrow \neg \mathit{rej}(\rho_4)\\
\sim tvon_2 &\leftarrow pfailure, \neg \mathit{rej}(\rho_5)\\
pfailure_2 &\leftarrow \neg \mathit{rej}(\rho_6)
\end{aligned}$$

$$\begin{aligned}
\mathit{rej}(\rho_1) &\leftarrow night, \neg tvon, \sim sleep_2\\
\mathit{rej}(\rho_2) &\leftarrow \sim night_2\\
\mathit{rej}(\rho_3) &\leftarrow tvon, \sim watchtv_2\\
\mathit{rej}(\rho_4) &\leftarrow \sim tvon_2
\end{aligned}$$

$$\begin{aligned}
sleep_1 &\leftarrow sleep_2 & \qquad sleep &\leftarrow sleep_1\\
night_1 &\leftarrow night_2 & \qquad night &\leftarrow night_1\\
tvon_1 &\leftarrow tvon_2 & \qquad tvon &\leftarrow tvon_1\\
watchtv_1 &\leftarrow watchtv_2 & \qquad watchtv &\leftarrow watchtv_1\\
\sim tvon_2 &\leftarrow \sim tvon_3 & \qquad \sim tvon &\leftarrow \sim tvon_2\\
pfailure_2 &\leftarrow pfailure_3 & \qquad pfailure &\leftarrow pfailure_2
\end{aligned}$$

whose unique answer set is

$$\{sleep_1, night, night_1, \mathsf{rej}(\rho_4), {\sim}tvon_2, pfailure, pfailure_2, sleep, {\sim}tvon\}$$

and its ***update answer*** *set is easily obtained:* $\{night, pfailure, sleep, {\sim}tvon\}$.

However, by following the original Definition 2 in [Eiter et al., 2002], the last four rules would not exist, for i should also equal n in Definition 4.1 item *(iv)*, and the answer set of the resulting program is

$$\{night, tvon, night_1, watchtv_1, tvon_1, pfailure_2, watchtv\}$$

that means the *TV is on* and the agent is watching it. On the other hand, by changing i to be within the range I suggest and by leaving the second rule in *(iv)* as the original definition, that is to say, $\ell \leftarrow \ell_1$, the resulting program would have the four rules

$$\begin{array}{ll} {\sim}tvon_2 \leftarrow {\sim}tvon_3 & {\sim}tvon \leftarrow {\sim}tvon_1 \\ pfailure_2 \leftarrow pfailure_3 & pfailure \leftarrow pfailure_1 \end{array}$$

instead, and would have the same strange answer set.

In other words, by following the original restriction presented in *(iv)* in Definition 2 in [Eiter et al., 2002], $pfailure \notin (\Pi_1, \Pi_2, \ldots, \Pi_n)$ when $1 \leq i < n$. Moreover, if the second rule in *(iv)* of Definition 4.1 was $\ell \leftarrow \ell_1$, a strange answer set would result:

$$\{night, night_1, pfailure_2, watchtv_1, watchtv, tvon_1, tvon\}$$

Anyway, their solver at `http://www.kr.tuwien.ac.at/staff/giuliana/project.html#Download` *seems*[1] to behave well. Unfortunately, their solver does not show intermediate transformations to figure out the correct semantic parameters so that I can give a precise statement. Notice that I have only provided a frond end to execute their solver in the web with a graphical interface, and I employed the latter as the main engine of my front end: `http://www2.in.tu-clausthal.de/~guadarrama/updates/upd.html`.

Back to the corrections I suppose, let us complete Observation 4.1:

Observation 4.2 (Continued from Observation 4.1)**.** *Consider again Observation 4.1 and perform a second update to the sequence with program* $\Pi_3 = \{{\sim}pfailure\}$. *Accordingly, the new answer set of the resulting update program is*

$$\{tvon_1, tvon, night_1, night, watchtv_1, watchtv, \mathsf{rej}(\rho_6), {\sim}pfailure_3, {\sim}pfailure\}$$

[1] Unfortunately the sources are not available so as to confirm the latest definition.

Finally, by Definition 4.2, the corresponding **update answer sets** are

$$\mathcal{U}(\Pi_1, \Pi_2) = \{night, pfailure, sleep, \sim tvon\}$$

and

$$\mathcal{U}(\Pi_1, \Pi_2, \Pi_3) = \{tvon, night, watchtv, \sim pfailure\}$$

Despite the complete deep nice analysis Eiter et al. make of known postulates and principles in the literature, one of the major shortcomings of their approach has to do with syntactic and semantic contents.

Take again, for instance, the example inspired from Alferes et al. and modified in Observation 1.1 that may produce *counterintuitive models*:

Observation 4.3. *[recapped from Observation 1.1]*

Suppose an agent who believes that when *it is* day *it is not* night *and vice versa, and that there are* stars *when it is* night *and when there are no* clouds*. Finally, that at the current moment it is a fact that there are no* stars*. This simple story may be coded*[1] *into* Π_1 *as follows:*

$$\begin{aligned} \Pi_1 = \{ day &\leftarrow \neg night \\ night &\leftarrow \neg day \\ stars &\leftarrow night, \neg cloudy \\ \sim stars &\leftarrow \top \} \end{aligned}$$

whose unique answer set is $\{day, \sim stars\}$*. Later, the agent acquires new information stating that* stars *and* constls *(*constellations*) are the same thing, as coded in* Π_2*. As soon as the agent updates* Π_1 *with program*

$$\begin{aligned} \Pi_2 = \{ stars &\leftarrow constls \\ constls &\leftarrow stars \} \end{aligned}$$

the expanded alphabet *of the two programs contains only one new* extra atom *with respect to* Π_1*: constls. As the model of* Π_2 *is obviously the empty answer set, constls is considered* synonym *of stars by means of* Π_2*, and thus the update should not change the* original beliefs*. However, the update yields an* extra answer set *in some of the existing update semantics based on the* causal rejection principle *—Section 4.2:*

$$\{stars, constls, night\}$$

[1] Notice that there are other ways to represent the story. The problem is, however, what to do in this particular situation, when the agent runs across this piece of information.

which does not coincide with common intuition.

The reason is that, although stars can not be true, introducing constls gives another possibility for stars to be true. Thus, the additional answer set is implied.

In general, these supplementary rules in the update are a conservative extension *[Osorio et al., 2001] to* Π_1*: the original language is extended and all answer sets ought to be extensions of the old answer sets. In this specific situation, constls should be true if and only if stars is true.*

To recapitulate, Eiter et al. were *very* good in gathering postulates and principles from the literature and in analysing them in terms of their proposal. Their approach, however, suffers from drawbacks owing to its reliance on the *causal rejection principle* [see Alferes et al., 1999].

4.2 DyLP and Other Dialects

One of the earliest approaches in updating logic programs appeared in late 90's in [Alferes et al., 1999, 1998] that was extended in an interesting language called LUPS by Alferes et al., to specify *explicit updates* in programs on a semantics that they called *Dynamic Logic Programming* or DyLP —[Alferes et al., 1999]. Some years later, they refined the latter in [Alferes et al., 2005], whom over the previous period formulated a *principle of rejection* (also *causal rejection principle* [see Alferes et al., 2005, 1999; Eiter et al., 2002] and [Alferes et al., 2005]) in the above citations.

Informally, the *refined principle* consists in *rejecting rules* of previous and upcoming programs in an *update sequence* whenever there are other rules at the current state with which they *conflict*.

Starting with motivation in Alferes et al., they claim to give a simple example to what they called a *tautology* (a rule from which they expect no models):

$$\neg p \leftarrow \neg p \tag{4.1}$$

Of course that rule alone does not produce any model in their semantics —i.e. just the **empty model**, {}. However, it is a clear counterexample why *strong negation* in our framework should not be a simple replacement to *"*$\neg\ell$*" in heads.*

Observation 4.4. *Take for example, program*

$$\{\sim p \leftarrow \neg p\}$$

whose unique ***answer set*** *is not the empty set. Namely, the answer set of the program is just* $\{\sim p\}$*.*

What is more, in their article, Alferes et al. explain in a footnote what *tautology* means: A rule of the form $\ell \leftarrow \mathsf{Body}$ with $\ell \in \mathsf{Body}$, where ℓ and Body are an atom

(or *default-negated atom*) and the body of a rule, respectively. This *high dependency on syntax* will prove to be one of their major shortcomings, as explained along this thesis.

Before starting with their proper definitions, a very special notation, taken from [Alferes et al., 2005], is necessary.

Let $\mathcal{A}$ be a set of propositional atoms. As before introduced, a *default literal* is an atom preceded by "$\neg$", while a *literal* is either an atom or a default literal. A *rule* ρ is an ordered pair $\mathsf{Head}(\rho) \leftarrow \mathsf{Body}(\rho)$ where $\mathsf{Head}(\rho)$ (the head of the rule) is a literal and $\mathsf{Body}(\rho)$ is a finite set of literals, and it has the form $\mathbf{L}_0 \leftarrow \mathbf{L}_1, \ldots, \mathbf{L}_n$. A rule with $\mathsf{Head}(\rho) = \mathbf{L}_0$ and $\mathsf{Body}(\rho) = \emptyset$ is called a fact, simply written as $\mathbf{L}_0$.

A *generalised logic program* (GLP[1]) Π over $\mathcal{A}$, is a finite or infinite set of rules, and $\Pi_\emptyset$ denotes an empty set of rules. If $\mathsf{Head}(\rho) = a$ (resp. $\mathsf{Head}(\rho) = \neg a$) then $\neg\mathsf{Head}(\rho) = \neg a$ (resp. $\neg\mathsf{Head}(\rho) = a$). Two rules ρ and ρ' are in *conflict*, denoted by themselves as $\rho \bowtie \rho'$, if and only if $\mathsf{Head}(\rho) = \neg\mathsf{Head}(\rho')$. An interpretation $\mathcal{M}$ of $\mathcal{A}$ is a set of atoms such that $\mathcal{M} \subseteq \mathcal{A}$. An atom a is true in $\mathcal{M}$, denoted by $\mathcal{M} \models a$, if and only if $a \in \mathcal{M}$, and false otherwise. A default literal $\neg a$ is true in $\mathcal{M}$, denoted by $\mathcal{M} \models \neg a$, if and only if $a \notin \mathcal{M}$, and false otherwise. A set of literals $\mathbf{L}$ is true in $\mathcal{M}$, denoted by $\mathcal{M} \models \mathbf{L}$, if and only if each literal in $\mathbf{L}$ is true in $\mathcal{M}$. A rule ρ is satisfied by an interpretation $\mathcal{M}$ if and only if whenever $\mathcal{M} \models \mathsf{Body}(\rho)$ then $\mathcal{M} \models \mathsf{Head}(\rho)$. An interpretation $\mathcal{M}$ is a model of a program Π if and only if $\mathcal{M}$ satisfies all rules in Π. An interpretation $\mathcal{M}$ of $\mathcal{A}$ is a stable model of a generalised logic program Π if and only if[2]

$$\overline{\mathcal{M}} = \mathsf{least}(\Pi \cup \{not\_a \mid a \notin \mathcal{M}\}) \tag{4.2}$$

where[3]

$$\overline{\mathcal{M}} = \mathcal{M} \cup \{not\_a \mid a \notin \mathcal{M}\} \tag{4.3}$$

with a as an atom and with $\mathsf{least}(\cdot)$ as the *least model* of the definite program obtained from the argument program by replacing every default literal $\neg a$ by a new atom not_a[4].

With this notation, one can define a *dynamic program* as follows.

Definition 4.3 (Dynamic Logic Program, DyLP [Alferes et al., 2005])**.** *A* dynamic logic program *(DyLP) is a sequence of generalised logic programs. Let* $\mathcal{P} = (\Pi_1, \ldots, \Pi_s)$ *and* $\mathcal{P}' = (\Pi'_1, \ldots, \Pi'_s)$ *be two DyLP's. The expression* $\rho(\Pi)$ *denotes the* set of all rules

[1] Although the name sounds familiar, GLP has little or nothing to do with generalised answer sets (GAS), from Section 3.6. Moreover, there is also an alternative to generalised logic programs called *Well Supported Semantics* [see Alferes et al., 2005].

[2] Consider that there seems to be a typo in [Alferes et al., 2005]: they typed "not a" rather than "not_a" in (4.2). Later they state that least function takes a *definite program* as an argument by replacing every *default literal* "not a" with a new atom not_a, which later results in a confusing abuse of notation with (4.6) and $\rho(\Pi)$.

[3] Notice that the authors did not define the domine of a in set $\overline{\mathcal{M}}$, which would result in an infinite set in (4.3).

[4] Note that indeed this sort of atoms is positive.

appearing in the programs $\Pi_1, \ldots, \Pi_s$, *and* $\mathcal{P} \cup \mathcal{P}'$ *denotes the* DyLP*:* $(\Pi_1 \cup \Pi'_1, \ldots, \Pi_s \cup \Pi'_s)$.

Intuitively, a *refined interpretation of a* DyLP program is a dynamic stable model if the least model(s) of the positive program that result(s) from the difference of the rejected rules and the union of the default assumptions is the same than the union of the interpretation and the default-negated literals that do not appear in the interpretation. The intuition behind $\mathsf{Rej}(\cdot,\cdot)$ is the set of rules that are in conflict with both current and previous rules in the sequence. Moreover, $\mathsf{Def}(\cdot,\cdot)$ consists of the positive "default-negated" atoms that do not appear in the intended model.

Definition 4.4 (Dynamic Stable Model [Alferes et al., 2005])**.** *Let* $\mathcal{P}$ *be a dynamic logic program and* $\mathcal{M}$ *an interpretation.* $\mathcal{M}$ *is a* dynamic stable model *of* $\mathcal{P}$ *if and only if*

$$\overline{\mathcal{M}} = \mathit{least}(\rho(\mathcal{P}) \setminus \mathit{Rej}(\mathcal{P}, \mathcal{M}) \cup \mathit{Def}(\mathcal{P}, \mathcal{M})) \tag{4.4}$$

where

$$\mathit{Rej}(\mathcal{P}, \mathcal{M}) = \{\rho \mid \rho \in \Pi_i, \exists \rho' \in \Pi_j, i \leq j, \rho \bowtie \rho', \mathcal{M} \models \mathit{Body}(\rho')\} \tag{4.5}$$

and[1]

$$\mathit{Def}(\mathcal{P}, \mathcal{M}) = \{not\_a \mid \nexists \rho \in \rho(\mathcal{P}), \mathit{Head}(\rho) = a, \mathcal{M} \models \mathit{Body}(\rho)\} \tag{4.6}$$

$\oplus_R$ *is the corresponding update operator.*

This approach has had several implementations for download, including one for the original version before the refined principle, and another for the *refined principle*. LUPS is also implemented and the following list shows their respective locations:

- `http://centria.di.fct.unl.pt/~jja/updates/dlp.html`
- `http://centria.di.fct.unl.pt/~banti/FedericoBantiHomepage/refdlp.htm`
- `http://centria.di.fct.unl.pt/~jja/updates/lups.html`

By considering Example 4.3 again, and inspired from the original example from Alferes et al., the reader may rewrite the pair of programs as follows.

[1] Notice that it seems they have missed the "_" and have typed "not a" rather than "not_a" in (4.6) from the original paper in [Alferes et al., 2005]. Moreover, (4.6) should be a set of rules, rather than a set of literals, to be sound with the equality in (4.4)! In the rest of the section, it is assumed the former, as the authors do.

Observation 4.5. *Let* $\mathcal{P} = \Pi_1 \oplus_R \Pi_2$*, where*

$$\begin{aligned} \Pi_1 = \{ day &\leftarrow \neg night \\ night &\leftarrow \neg day \\ stars &\leftarrow night, \neg cloudy \\ not\_stars &\leftarrow \top \} \end{aligned}$$

and

$$\begin{aligned} \Pi_2 = \{ stars &\leftarrow constls \\ constls &\leftarrow stars \} \end{aligned}$$

A resulting *dynamic stable model is* $\mathcal{M} = \{day\}$ *because* **least** *function of the definite logic program gives out the following model*

$$\{day, not\_night, not\_cloudy, not\_stars, not\_constls\}$$

where

$$\mathit{Rej}(\mathcal{P}, \mathcal{M}) = \{stars \leftarrow night, \neg cloudy\}$$

and

$$\mathit{Def}(\mathcal{P}, \mathcal{M}) = \{not\_night, not\_stars, not\_cloudy, not\_constls\}$$

Thus, $\overline{\mathcal{M}} = \mathcal{M} \cup \{not\_night, not\_stars, not\_cloudy, not\_constls\}$*, which equals the model got by* **least**. *Moreover, the interpretation* $\mathcal{M} = \{night, stars, constls\}$ ***is not*** *a refined dynamic stable model, because* $\mathit{Def}(\mathcal{P}, \mathcal{M}) = \{not\_cloudy, not\_day\}$ *and* $\mathit{Rej}(\mathcal{P}, \mathcal{M}) = \{(stars \leftarrow night, \neg cloudy), (not\_stars \leftarrow \top)\}$*. As a result, the* **least** *model*

$$\{night, not\_day, not\_cloudy\} \neq \overline{\mathcal{M}}$$

which means $\mathcal{M}$ *is not a refined dynamic stable model, as one would expect, despite the unnecessary emulation of strong negation and "default" negation in heads.*

Next, let us analyse an example inspired by Sakama and Inoue, originally proposed by Alferes et al..

Observation 4.6. *Suppose the same story introduced in Observations 4.1 and 4.2 as*

follows: $\mathcal{P} = \Pi_1 \oplus_R \Pi_2 \oplus_R \Pi_3$ *where*

$$\begin{aligned} \Pi_1 = \{sleep &\leftarrow \neg tvon \\ watchtv &\leftarrow tvon \\ tvon &\leftarrow \top \\ \Pi_2 = \{pfailure &\leftarrow \top \\ not\_tvon &\leftarrow pfailure\} \\ \Pi_3 = \{\neg pfailure &\leftarrow \top\} \end{aligned}$$

$\mathcal{M} = \{tvon, watchtv\}$ *is a refined dynamic stable model of the update, since the unique rejected rule is* $pfailure \leftarrow \top$ *and the unique default is* not_sleep*, where the* **least** *model is* $\{watchtv, tvon, not\_pfailure, not\_sleep\} = \overline{\mathcal{M}}$*. However, Sakama and Inoue argue that the resulting model is* ***counterintuitive*** *because, after the first update, the refined dynamic stable model was* $\{pfailure, sleep\}$ *and thus there is no reason to believe that once the electric power is restored, the TV should be on back again! A similar counter-intuitive result is given in Observation 4.2.*

Despite its nice behaviour in those typical examples up to now analysed, there is still a simple example, first suggested by Eiter et al., that still causes counterintuitive results in this DyLP-semantics.

Observation 4.7. *Suppose an initial knowledge base* $\Pi_0 = \{(c \leftarrow r), (r \leftarrow \top)\}$ *updated with* $\Pi_1 = \{\text{not}\, r \leftarrow \text{not}\, c\}$*. Firstly, the initial generalised stable* ***model*** *of* Π_0 *is* $\{c, r\}$*, and one would expect no changes after the update. However, the update* $\mathcal{P} = \Pi_0 \oplus_R \Pi_1$ *has the* ***extra model*** $\mathcal{M} = \{not\_c, not\_r\}$ *because* $\overline{\mathcal{M}} = \{not\_c, not\_r\}$*;* $Rej(\mathcal{P}, \mathcal{M}) = \{r \leftarrow \top\}$*;* $Def(\mathcal{P}, \mathcal{M}) = \{not\_c\}$*; and* $least[(\Pi_0 \cup \Pi_1) \setminus Rej(\mathcal{P}, \mathcal{M}) \cup Def(\mathcal{P}, \mathcal{M})] = \{not\_c, not\_r\} = \overline{\mathcal{M}}$*.*

Although it is true that Alferes et al. were some of the first researchers [Alferes et al., 1999] to formulate and implement a semantics for updates, one can easily realise the clear shortcomings this approach has: firstly for the different syntax of the so-called generalised logic programs that is a different concept of SM-semantics —a *non-standard concept of SM* [Eiter et al., 2002]; secondly for their *causal rejection principle* that produces the mentioned *counterintuitive results* and excludes the approach, as they themselves state it in [Alferes et al., 2005], from other semantics like Sakama and Inoue's or Zhang and Foo's —[see Alferes et al., 2005].

4.3 Sakama & Inoue

Sakama and Inoue; Sakama and Inoue propose *three types of updates*, to which they call

inconsistency removal, *view updates* and *theory updates*. Each of them correspond to a special case of updates and revision in the literature.

In particular, my thesis is focused on theory updates, rather than other special cases of making an inconsistent program consistent or differentiating between variant and *invariant knowledge*. As a result, this section does not describe the other types here, although they are sometimes implicit in the proposals of this thesis, like for example a special case of *recovering consistency* from a program or sequence of programs. They are studied in upcoming sections.

4.3.1 Extended Abduction Framework

Before introducing the definitions for theory updates, some new notation and specialised terminology is necessary to understand their approach. For instance, the authors define their particular framework of abduction, and they call it *extended abduction*, which differs from the standard abduction framework in [Kakas et al., 1998; Kakas and Mancarella, 1990; Poole, 1988]. For instance, besides an *explanation* to satisfy

$$K \cup E \models G$$

they also introduce the notion of *negative explanations*, such that

$$K \setminus F \models G$$

where K is a first-order theory; E, F sets of hypotheses; G an observation; and $K \cup E$ and $K \setminus F$ are consistent.

According to Sakama and Inoue, an *extended abductive program* is a pair $\langle P, \mathcal{A}^\bullet \rangle$, where P and $\mathcal{A}^\bullet$ are DLP's. An extended abductive program $\langle P, \mathcal{A}^\bullet \rangle$ is *consistent* if P is consistent.

In the process of updating a program with another, Sakama and Inoue define a set of conditions that the intended update must meet.

Definition 4.5 (Theory Updates [Sakama and Inoue, 2003])**.** *Given a DLP-program pair P and Q, P' accomplishes a* theory update *of P by Q if*

1. *P' is consistent,*

2. *$Q \subseteq P' \subseteq P \cup Q$,*

3. *there is no consistent program P'' such that $P' \subset P'' \subseteq P \cup Q$.*

In words of Sakama and Inoue, the intended update is the union of the new information and a maximal subset of the original program that is consistent with the update, which obviously is not always unique.

Their update approach starts with the extended abductive program

$$\langle\, P \cup Q, P \setminus Q \,\rangle$$

The intuition behind this program consists in merging the update with the original theory and combining the rules of P that do not belong to Q so as to get a consistent update.

In order to reduce the set of abducible rules $P \setminus Q$ to abductive facts and to compute their models in a conventional way, the extended abductive program (as defined in [Inoue and Sakama, 1995]) must be transformed into a *normal* (traditional) abductive program —similar to [Kakas et al., 1998]— in which abducibles contain only non-disjunctive facts, as in the definition below. In this manner, the models of an update program (later introduced) will contain both facts and names of rules to remove, rather than the rules themselves.

Definition 4.6 (Normalised Abductive Program [Sakama and Inoue, 2003])**.** *Given an extended abductive program* $\langle\, P, \mathcal{A}^\bullet \,\rangle$*, and*

$$\mathcal{R} = \{\Sigma \leftarrow \Gamma \mid (\Sigma \leftarrow \Gamma) \in \mathcal{A}^\bullet \text{ and } \Sigma \leftarrow \Gamma \text{ is not a non-disjunctive fact}\}.$$

Then, let

$$\begin{aligned} P^{\mathrm{n}} &= (P \setminus \mathcal{R}) \cup \{\Sigma \leftarrow \Gamma, \gamma_R \mid R = (\Sigma \leftarrow \Gamma) \in \mathcal{R}\} \qquad (4.7)\\ &\quad \cup \{\gamma_R \leftarrow \top \mid R \in \mathcal{R} \cap P\},\\ \mathcal{A}^{\bullet \mathrm{n}} &= (\mathcal{A}^\bullet \setminus \mathcal{R}) \cup \{\gamma_R \leftarrow \top \mid R \in \mathcal{R}\} \qquad (4.8) \end{aligned}$$

where γ_R *is a newly introduced atom (called the* name of R*) uniquely associated with each rule* R *in* $\mathcal{R}$*, and* $\langle\, P^{\mathrm{n}}, \mathcal{A}^{\bullet \mathrm{n}} \,\rangle$ *is the normalised form of* $\langle\, P, \mathcal{A}^\bullet \,\rangle$*. For any rule* $R \in \mathcal{R}$*, its name comes from the function* $n(R) = \gamma_R$*. In particular, any abducible fact* $L \leftarrow \top$ *has the name* L*, i.e.,* $n(L) = L$*.*

Note that in [Sakama and Inoue, 2003], there seems to be a typo when they wrote (4.8) as $\mathcal{A}^{\bullet \mathrm{n}} = (\mathcal{A}^\bullet \setminus \mathcal{R}) \cup \{\gamma_R \mid R \in \mathcal{R}\}$.

4.3.2 $\diamond_{\mathsf{SI}}$-operation

Once the extended abductive program is normalised, its interpretations shall correspond to *update programs* that consist of the rules of the original theory that don't belong to the *normalised abductive set*, merged with a new set of update rules, as following specified.

Definition 4.7 (Update Rule; Update Atom [Sakama and Inoue, 2003])**.** *Given an abductive program* $\langle\, P, \mathcal{A}^\bullet \,\rangle$*, where* $\mathcal{A}^\bullet$ *contains only (non-disjunctive) facts, the set* $\mathcal{UR}$ *of* update rules *is constructed as follows.*

1. *For any literal*[1] $a \in \mathcal{A}^\bullet$*, the following rules are in* $\mathcal{UR}$*:*

$$a \leftarrow \neg\overline{a},$$
$$\overline{a} \leftarrow \neg a,$$

where $\overline{a}$ *is a newly introduced atom uniquely associated with* a*. The above pair of rules is* $abd(a)$ *hereafter. In addition, yet another semantically equivalent way to represent it, according to Sakama and Inoue, is by* $a \vee \overline{a} \leftarrow \top$*.*

2. *For any literal* $a \in \mathcal{A}^\bullet \setminus P$*, the following rule is in* $\mathcal{UR}$*:*

$$a^+ \leftarrow a.$$

3. *For any literal* $a \in \mathcal{A}^\bullet \cap P$*, the following rule is in* $\mathcal{UR}$*:*

$$a^- \leftarrow \neg a.$$

where a^+ *and* a^- *are atoms uniquely associated with any* $a \in \mathcal{A}^\bullet$*, which they call* update atoms*.*

Sakama and Inoue interpret, at a *meta-level*, that a^+ means making a true, when it is not in P, while a^- means making a false when it is in P. In other words, they represent the introduction and deletion of a, respectively. On the other hand, $\overline{a}$ would represent an *unknown truth value* of a, i.e. neither true nor false a. Last, they define the set of all *update atoms* associated with the abducibles in $\mathcal{A}^\bullet$ by $\mathcal{UA}$. That is to say, $\mathcal{UA} = \mathcal{UA}^+ \cup \mathcal{UA}^-$, where $\mathcal{UA}^+$ and $\mathcal{UA}^-$ are the sets of update atoms of form a^+ and a^-, respectively.

Next, these *update rules* take part of the update program of the normalised extended abductive program that is an intermediate EDLP. This intermediate program specification is as follows:

Definition 4.8 (Update Program, $\mathcal{UP}$ [Sakama and Inoue, 2003])**.** *Given an abductive program* $\langle P, \mathcal{A}^\bullet \rangle$*, its* update program*,* $\mathcal{UP}$*, is defined as an EDLP such that*

$$\mathcal{UP} = (P \setminus \mathcal{A}^\bullet) \cup \mathcal{UR}.$$

[1]Note that it seems there is an ambiguity here, as in Definition 4.6, the authors construct $\mathcal{A}^{\bullet n}$ with *literals* unified to *rules*! Then, in Definition 4.7 they state that "for every literal in $\mathcal{A}^\bullet \ldots$", which confirms that $\mathcal{A}^\bullet$ is already normalised, but that it contains *literals* rather than facts. So, strictly speaking, I would use the term signature of $\mathcal{A}^\bullet$ rather than $\mathcal{A}^\bullet$ alone. Moreover, I would change the construction of $\mathcal{A}^{\bullet n}$ to $\mathcal{A}^{\bullet n} = (\mathcal{A}^\bullet \setminus \mathcal{R}) \cup \{\gamma_R \leftarrow \top \mid R \in \mathcal{R}\}$

Then the models of an *update program* denote the deletion of facts or rules from the original program in the pair. As a result, one or more new updated programs are constructed.

Definition 4.9 (U-minimal Answer Sets [Sakama and Inoue, 2003])**.** *An answer set $\mathcal{S}$ of $\mathcal{UP}$ is called* U-minimal *(U-MAS) if there is no answer set $\mathcal{S}'$ of $\mathcal{UP}$ such that $\mathcal{S}' \cap \mathcal{UA} \subset \mathcal{S} \cap \mathcal{UA}$.*

4.3.3 Discussion

This abduction framework proves to have nice properties of a *syntactical minimal change* of rules in the original non-monotonic theory (Definition 4.5) by means of consistent interpretations of *hypothetical changes*. With this framework, they can perform particular kinds of updates and maintenance of their *consistency*, by providing a vast analysis of shortcomings in other approaches.

Sakama and Inoue's first goal is providing a mechanism (an update semantics) to compute their extended abduction. Secondly, they also characterise updates through extended abduction, as they themselves state it. Consequently, the approach lacks of a proper analysis of more principles and postulates from the literature. Additionally, they characterise different kinds of updates with their extended abduction, claiming that they can provide an algebra of rules deletion, besides the addition of them, to explain observations.

Let us recapitulate Sakama and Inoue's approach in a few words. They construct their *update program* out of the normal abductive form of an *extended abductive program*

$$\langle\, P \cup Q, P \setminus Q \,\rangle$$

whose models are *U-MAS's*, interpreted from an update program. Last, the interpretation produces one (or more) new programs representing knowledge bases, derived from the addition/deletion of facts that the U-MAS's describe in turn.

In order to illustrate the above definitions, consider the following example extended from the original in [Sakama and Inoue, 2003][1] that shows one of the differences with several approaches.

Observation 4.8. *Suppose an update to the knowledge base*

$$\begin{aligned} \Pi_1 = \{ sleep &\leftarrow \neg tvon \\ watchtv &\leftarrow tvon \\ tvon &\leftarrow \top \} \end{aligned}$$

[1] Originally, Alferes et al. proposed this example, but it is a little modified in [Sakama and Inoue, 2003] to contrast their differences. Moreover, I have extended it here in order to see further details.

with[1]

$$\Pi_2 = \{ pfailure \leftarrow \top \quad \bot \leftarrow pfailure, tvon \}$$

The situation is to be coded into the extended abductive program $\langle\, \Pi_1 \cup \Pi_2, \Pi_1 \setminus \Pi_2 \,\rangle$. *The update program* $\mathcal{UP}$ *of* $\langle\, (\Pi_1 \cup \Pi_2)^{\mathrm{n}}, (\Pi_1 \setminus \Pi_2)^{\mathrm{n}} \,\rangle$ *is specified as*

$$\begin{aligned} \mathcal{UP}: \quad & pfailure \leftarrow \top \\ & \bot \leftarrow pfailure, tvon \\ & sleep \leftarrow \neg tvon, \gamma_1 \\ & watchtv \leftarrow tvon, \gamma_2 \\ & abd(tvon), abd(\gamma_1), abd(\gamma_2), \\ & tvon^{-} \leftarrow \neg tvon \\ & \gamma_1^{-} \leftarrow \neg\gamma_1 \\ & \gamma_2^{-} \leftarrow \neg\gamma_2 \end{aligned}$$

where γ_1 *and* γ_2 *are names of the abducible rules in* $\Pi_1 \setminus \Pi_2$. *Then,* $\mathcal{UP}$ *has the unique U-MAS*

$$\{ pfailure, sleep, \overline{tvon}, tvon^{-}, \gamma_1, \gamma_2 \}$$

which represents the deletion of fact tvon from $\Pi_1 \cup \Pi_2$. *Accordingly, the update of* Π_2 *to* Π_1 *is the resulting program*

$$\begin{aligned} \Pi_3: \quad & sleep \leftarrow \neg tvon \\ & watchtv \leftarrow tvon \\ & pfailure \leftarrow \top \\ & \bot \leftarrow pfailure, tvon \end{aligned}$$

whose ***answer set*** *is just* $\{pfailure, sleep\}$

Next, suppose ***yet another update*** $(\Pi_4 : \neg pfailure \leftarrow \top)$ *to* Π_3, *which represents that there is power again. Sakama and Inoue code this new pair by the abductive program*

[1] In [Alferes et al., 1999] the rule "$\leftarrow pfailure, tvon$" is given as "$\neg tvon \leftarrow pfailure$". These two rules are semantically equivalent under the answer set semantics, as the authors explain in [Sakama and Inoue, 2003]. However, as later seen in this example, the difference between either expression would result in the existence of $\sim tvon$ in the corresponding model!

$\langle\, \Pi_3 \cup \Pi_4, \Pi_3 \setminus \Pi_4 \,\rangle$. *So, the update program of* $\langle\, (\Pi_3 \cup \Pi_4)^{\mathrm{n}}, (\Pi_3 \setminus \Pi_4)^{\mathrm{n}} \,\rangle$ *turns into*

$$\begin{aligned}
\mathcal{UP}: \quad & \neg pfailure \leftarrow \top \\
& sleep \leftarrow \neg tvon\gamma_1 \\
& watchtv \leftarrow tvon\gamma_2 \\
& \bot \leftarrow pfailure, tvon\gamma_3 \\
& abd(pfailure), abd(\gamma_1), abd(\gamma_2), abd(\gamma_3), \\
& pfailure^{-} \leftarrow \neg pfailure \\
& \sim\!\gamma_1 \leftarrow \neg\gamma_1, \sim\!\gamma_2 \leftarrow \neg\gamma_2, \sim\!\gamma_3 \leftarrow \neg\gamma_3.
\end{aligned}$$

Then, $\mathcal{UP}$ *has the unique* U-MAS

$$\{\neg pfailure, sleep, \gamma_1, \gamma_2, \gamma_3, \overline{pfailure}, \sim\!pfailure\},$$

which implies that the result of the update is $(\Pi_3 \cup \Pi_4) \setminus \{pfailure \leftarrow \top\}$. *As a result, the* ***unique answer set*** *of the unique resulting program is* $\{\sim\!pfailure, sleep\}$.

Sakama and Inoue propose this example as an argument against other approaches like [Alferes et al., 1999; Eiter et al., 2002][1], which bring back previous knowledge of the original theory. That is to say, their interpretation is that the TV turns itself on again and it is possible to watch it as well: $\{tvon, watchtv, \sim\!pfailure\}$, which does not coincide with their intuition. However, this argument seems to be too strong to generalise that all update semantics should behave accordingly, because Sakama and Inoue are differentiating *fluents* and *actions* in a language that does not have such an explicit difference.

In order to illustrate this assumption, let us modify Example 4.8 in such a way that the language contains only fluents, naturally at a higher abstraction level. As a result, the new story goes like this.

Observation 4.9. *Suppose a learning agent whose simple knowledge base states that he is* innocent *when he is not* guilty*, and at the beginning he believes he is not guilty, thus innocent. A following update states that any person is no longer innocent when guilty, that the person is guilty when* murderer *and now the agent in question turns out to be a murderer! Thus, he is no longer innocent. However, more relevant rules come up that state that any person is not a murderer when* self defended*; that the person is self defended when first attacked; and it is a fact that the agent in question was first attacked. Consequently, common intuition would suggest that the agent's innocence should be in*

[1] This is also the case for our approach in [Guadarrama et al., 2006].

effect. Under Sakama and Inoue's approach, however, that previous knowledge *is lost forever and there is no way to conclude that he is* innocent*.*

Here is how Sakama and Inoue's approach can process this knowledge. The knowledge base consists of the original theory represented by Π_1*, as well as an update to it,* Π_2*, where*

$$\Pi_1 : \{ \begin{array}{rcl} \mathit{innocent} & \leftarrow & \sim \mathit{guilty} \\ \sim \mathit{guilty} & \leftarrow & \top \} \end{array} \qquad \Pi_2 : \{ \begin{array}{rcl} \sim \mathit{innocent} & \leftarrow & \mathit{guilty} \\ \mathit{guilty} & \leftarrow & \mathit{murderer} \\ \mathit{murderer} & \leftarrow & \top \} \end{array}$$

and its normalised abductive program $\langle\, P^n, \mathcal{A}^{\bullet n} \,\rangle$*, where*

$$P^n = \{ \begin{array}{rcl} \sim \mathit{guilty} & \leftarrow & \top \\ \gamma_1 & \leftarrow & \top \\ \mathit{murderer} & \leftarrow & \top \\ \mathit{innocent} & \leftarrow & \sim \mathit{guilty}, \gamma_1 \\ \sim \mathit{innocent} & \leftarrow & \mathit{guilty} \\ \mathit{guilty} & \leftarrow & \mathit{murderer} \} \end{array} \qquad \mathcal{A}^{\bullet n} = \{ \begin{array}{rcl} \sim \mathit{guilty} & \leftarrow & \top \\ \gamma_1 & \leftarrow & \top \} \end{array}$$

The update rules and the update program consist respectively of

$$UR : \{ \begin{array}{rcl} \sim \mathit{guilty} \vee \overline{\sim \mathit{guilty}} & \leftarrow & \top \\ \sim \mathit{guilty}^- & \leftarrow & \neg \sim \mathit{guilty} \\ \gamma_1 \vee \overline{\gamma_1} & \leftarrow & \top \\ \gamma_1^- & \leftarrow & \neg \gamma_1 \} \end{array} \qquad \mathcal{UP} : \{ \begin{array}{rcl} \sim \mathit{guilty} \vee \overline{\sim \mathit{guilty}} & \leftarrow & \top \\ \sim \mathit{guilty}^- & \leftarrow & \neg \sim \mathit{guilty} \\ \gamma_1 \vee \overline{\gamma_1} & \leftarrow & \top \\ \gamma_1^- & \leftarrow & \neg \gamma_1 \\ \mathit{murderer} & \leftarrow & \top \\ \mathit{innocent} & \leftarrow & \sim \mathit{guilty}, \gamma_1 \\ \sim \mathit{innocent} & \leftarrow & \mathit{guilty} \\ \mathit{guilty} & \leftarrow & \mathit{murderer} \} \end{array}$$

that has two answer sets

$$\{murderer, \overline{\sim guilty}, \sim guilty^{-}, \overline{\gamma_1}, \gamma_1^{-}, \sim innocent, guilty\}$$
$$\{murderer, \overline{\sim guilty}, \sim guilty^{-}, \gamma_1, \sim innocent, guilty\}$$

from which the unique U-MAS

$$\{murderer, \overline{\sim guilty}, \sim guilty^{-}, \gamma_1, \sim innocent, guilty\}$$

leads to the ***updated knowledge base*** *where* $\sim guilty$ *is no longer present:*

$$\begin{aligned} \Pi_3 : \{ innocent &\leftarrow \sim guilty \\ \sim innocent &\leftarrow guilty \\ guilty &\leftarrow murderer \\ murderer &\leftarrow \top \} \end{aligned}$$

Next, the following program represents the second update as:

$$\begin{aligned} \Pi_4 : \{ \sim murderer &\leftarrow self\_defence \\ self\_defence &\leftarrow attacked \\ attacked &\leftarrow \top \} \end{aligned}$$

which, after the same process yields the following update program

$$\begin{aligned}
\mathcal{UP} : \{ \gamma_1 \vee \overline{\gamma_1} &\leftarrow \top & \gamma_3^{-} &\leftarrow \neg\gamma_3 \\
\gamma_1^{-} &\leftarrow \neg\gamma_1 & attacked &\leftarrow \top \\
\gamma_2 \vee \overline{\gamma_2} &\leftarrow \top & innocent &\leftarrow \sim guilty, \gamma_1 \\
\gamma_2^{-} &\leftarrow \neg\gamma_2 & \sim innocent &\leftarrow guilty, \gamma_2 \\
murderer \vee \overline{murderer} &\leftarrow \top & guilty &\leftarrow murderer, \gamma_3 \\
murderer^{-} &\leftarrow \neg murderer & \sim murderer &\leftarrow self_defence \\
\gamma_3 \vee \overline{\gamma_3} &\leftarrow \top & self_defence &\leftarrow attacked \}
\end{aligned}$$

that has the following answer sets

$$\{attacked, \overline{\gamma}_1, \gamma_1^-, \overline{\gamma}_2, \gamma_2^-, \overline{murderer}, murderer^-, \overline{\gamma}_3, \gamma_3^-, \sim murderer, self\_defence\}$$
$$\{attacked, \gamma_1, \overline{\gamma}_2, \gamma_2^-, \overline{murderer}, murderer^-, \overline{\gamma}_3, \gamma_3^-, \sim murderer, self\_defence\}$$
$$\{attacked, \overline{\gamma}_1, \gamma_1^-, \gamma_2, \overline{murderer}, murderer^-, \overline{\gamma}_3, \gamma_3^-, \sim murderer, self\_defence\}$$
$$\{attacked, \gamma_1, \gamma_2, \overline{murderer}, murderer^-, \overline{\gamma}_3, \gamma_3^-, \sim murderer, self\_defence\}$$
$$\{attacked, \overline{\gamma}_1, \gamma_1^-, \overline{\gamma}_2, \gamma_2^-, \overline{murderer}, murderer^-, \gamma_3, \sim murderer, self\_defence\}$$
$$\{attacked, \gamma_1, \overline{\gamma}_2, \gamma_2^-, \overline{murderer}, murderer^-, \gamma_3, \sim murderer, self\_defence\}$$
$$\{attacked, \overline{\gamma}_1, \gamma_1^-, \gamma_2, \overline{murderer}, murderer^-, \gamma_3, \sim murderer, self\_defence\}$$
$$\{attacked, \gamma_1, \gamma_2, \overline{murderer}, murderer^-, \gamma_3, \sim murderer, self\_defence\}$$

with the unique U-MAS

$$\{attacked, \gamma_1, \gamma_2, \overline{murderer}, murderer^-, \gamma_3, \sim murderer, self\_defence\}$$

that produces a knowledge base

$$\begin{aligned} P_5 : \{innocent &\leftarrow \sim guilty \\ \sim innocent &\leftarrow guilty \\ guilty &\leftarrow murderer \\ \sim murderer &\leftarrow self\_defence \\ self\_defence &\leftarrow attacked \\ attacked &\leftarrow \top\} \end{aligned}$$

whose ***model*** $\{attacked, \sim murderer, self\_defence\}$ *reflects the* loss of previous relevant information *—no conclusions about guilt or innocence are available.*

If this *counterintuitive example* was not enough, let us change a bit the original Example 4.8 in such a way that both actions and fluents are inverted.

Observation 4.10. *Suppose a simple scenario where an agent can see in a room and the blinds of the room are open. Later, new information is at hand and the agent knows that it cannot see when the blinds are closed; that by closing them means they are closed, and that they cannot be closed and open at the same time. Simultaneously, there is also an event that closes the blinds. Following, a program that codes the initial information:*

updated with

$$\Pi_1 : \{can\_see \leftarrow b\_open \quad b\_open \leftarrow \top\} \qquad \Pi_2 : \{\sim can\_see \leftarrow b\_closed \quad \bot \leftarrow b\_open, b\_closed \quad b\_closed \leftarrow close\_b \quad close\_b \leftarrow \top\}$$

After updating Π_1 *with* Π_2*, the update program*

$$\begin{aligned}
\mathcal{UP} : \{b_open \vee \overline{b_open} &\leftarrow \top \\
b_open^- &\leftarrow \neg b_open \\
\gamma_1 \vee \overline{\gamma_1} &\leftarrow \top \\
\gamma_1^- &\leftarrow \neg\gamma_1 \\
close_b &\leftarrow \top \\
can_see &\leftarrow b_open, \gamma_1 \\
\sim can_see &\leftarrow b_closed \\
\bot &\leftarrow b_open, b_closed \\
b_closed &\leftarrow close_b\}
\end{aligned}$$

has the following U-MAS:

$$\{close\_b, \overline{b\_open}, b\_open^-, \gamma_1, \sim can\_see, b\_closed\}$$

This model means the deletion of fact b_open *from the original knowledge base.*

Now suppose the agent decides not to close the blinds when it is reading, that it is reading when it wants to read, and that now it wants to read. Then, the updated program and the new update are

$$\begin{aligned}
\{can_see &\leftarrow b_open \\
\sim can_see &\leftarrow b_closed \\
\bot &\leftarrow b_open, b_closed \\
b_closed &\leftarrow close_b \\
close_b &\leftarrow \top\}
\end{aligned}
\qquad
\begin{aligned}
\{\sim close_b &\leftarrow reading \\
reading &\leftarrow want_to_r \\
want_to_r &\leftarrow \top\}
\end{aligned}$$

whose unique U-MAS

$$\{want\_to\_r, \gamma_1, \gamma_2, \gamma_3, \overline{close\_b}, close\_b^{-}, \gamma_4, {\sim}close\_b, reading\}$$

produces an updated program

$$\begin{aligned}
\{can_see &\leftarrow b_open\\
{\sim}can_see &\leftarrow b_closed\\
\bot &\leftarrow b_open, b_closed\\
b_closed &\leftarrow close_b\\
{\sim}close_b &\leftarrow reading\\
reading &\leftarrow want_to_r\\
want_to_r &\leftarrow \top\}
\end{aligned}$$

with an ***answer set*** *that again reflects a* loss of information *on the ability to see:*

$$\{want\_to\_r, {\sim}close\_b, reading\}$$

Clearly, the objection Sakama and Inoue propose against other semantics may have different interpretations in *planning scenarios*, where there is indeed a formal explicit distinction between *fluents and actions*, and might be a matter for *further investigation.* Meanwhile, neither interpretation is correct or incorrect when talking about simple logic-program updates, unless formalising which rules must persist and which must not, which is clearly beyond the scope of my thesis.

Finally, there is a simple example that might represent another disadvantage of this approach.

Observation 4.11. *Suppose the initial knowledge base* $\Pi = \emptyset$ *updated by a simple fact* $\Pi_1 = \{x \leftarrow \top\}$. *Following Sakama and Inoue's framework, the answer sets of its update program is empty:* $\mathcal{UP} = (\emptyset \setminus \{x \leftarrow \top\}) \cup \mathcal{UR}$, *where* $\mathcal{UR}$ *is clearly empty because the extended abductive program from the update pair has no abducibles:* $\mathcal{A}^{\bullet} = \emptyset \setminus \{x \leftarrow \top\}$.

Moreover, although the authors present a deep analysis of their proposal and although it seems to be *robust*-enough for agent's *changing environment*, there is a lack of further and more *general properties* and solver that make it hard to compare with other alternative approaches. As pointed out by Zhang, this approach is classified into a *syntax-based semantics.* As a result, it has no general semantic foundation that justifies its updates [Zhang, 2006], and by interpreting the resulting knowledge bases with a given semantics might interfere with the ASP semantics that performs the update operation. It is clear that they justify their updates with an *extended abductive frame-*

work, which is still a specific problem and then leaves the mentioned absence of update characterisation.

Finally, a minor disadvantage is that the approach is undefined to update a knowledge base with an inconsistency. Sakama and Inoue state that such a kind of update "makes no sense" in Definition 4.5, which clearly does not mean that an agent will *never* come across an originally *inconsistent observation*. Nevertheless, they do consider cases where an *initial knowledge base* is originally inconsistent, and they identify such case as *inconsistency removal*. This method consists in updating an inconsistency knowledge base with an *empty update*.

4.4 Zhang's line

An interesting proposal for updates comes from Zhang that followed preliminary proposals reported in [Zhang, 1995; Zhang and Foo, 1998], where the author identifies three types of problems to solve in an update process: *elimination of contradictory information*, *conflict resolution* and *syntactic representation*.

Additionally, one of the applications from that line is an interesting language introduced in [Crescini and Zhang, 2005] that is specialised in updates of agent *policies* and defined at the top of ASP. Crescini and Zhang specify such policies in terms of clauses with a predefined semi-imperative *syntactical structure*, as well as an initial *planning approach*.

However, owing to a special focus the work has on *policies*, the programmer is restricted and obliged to use *reserved words* like "always", "implied by", "with absence", etc. which, besides constraining the domain to specific applications, it 'reduces' the language and has potentially different meanings in the *meta-language*. Nevertheless, they already have a fully-fledged system, as they themselves mention it in [Crescini and Zhang, 2005].

4.4.1 General View

As mentioned above, Zhang characterises updates in terms of three main objectives: *contradiction elimination*, *conflict resolution* and *syntactic representation*. The first topic is one of the most obvious in semantics for updates, which should be real by preserving a *minimal-change principle* and a proper *justification*. On the other hand, *conflict resolution* has to do with potential future *contradictions* an update might yield because of the introduction of the two kinds of negations in logic programs —strong and default negation. Finally, once the process meets the two main goals, the author argues that a proper semantics should also *preserve* as many as possible of the original rules from the *updating knowledge base*.

In order to realise these three goals, Zhang characterises a program update by means of a called *prioritised logic program*. In an intuitive way, this kind of program consists in *preferring the latest update* to the *original knowledge base* including non-contradictory but *conflicting rules*.

Zhang motivates his proposal by introducing a representative example that exposes the two kinds of problems he studied, and the example looks as follows:

Observation 4.12 ([Zhang and Foo, 2005])**.** *Suppose*

$$\begin{aligned}\Pi_0 = \{member(a,g) &\leftarrow \top\\ member(b,g) &\leftarrow \top\\ access(a,f_2) &\leftarrow \top\\ access(X,f_1) &\leftarrow member(X,g)\\ {\sim}access(X,f_2) &\leftarrow member(X,g), \neg access(X,f_2)\}\end{aligned}$$

updated with

$$\begin{aligned}\Pi_1 = \{member(c,g) &\leftarrow \top\\ {\sim}access(X,f_1) &\leftarrow member(X,g)\\ access(X,f_2) &\leftarrow member(X,g), \neg {\sim}access(X,f_2)\}\end{aligned}$$

According to Zhang, this update ought to have the ***unique answer set***

$$\begin{aligned}S = \{&{\sim}access(a,f_1), {\sim}access(b,f_1), {\sim}access(c,f_1),\\ &access(a,f_2), access(b,f_2), access(c,f_2)\}\end{aligned}$$

Then he claims that the last rule in Π_1,

$$access(X,f_2) \leftarrow member(X,g), \neg {\sim}access(X,f_2)\}$$

should *override* rule

$${\sim}access(X,f_2) \leftarrow member(X,g), \neg access(X,f_2)\}$$

in Π_0. That is to say, Zhang states that there is information loss in some other preliminary semantics, but at the same time, his semantics says both b and c have access to f_2, ignoring the possible situation when they explicitly do not. In fact, one might expect that b has no access to f_2 in Π_0 and that such a situation persists.

Regarding the controversy from this syntactical change of rule, his approach proposes a two-fold process of *eliminating contradictory information*, as well as *resolution of conflicting rules* and a final *syntactic representation* stage. Nevertheless, before the introduction of those two main processes, some fundamental definitions are in order.

The following definition can be seen as assuming true the given ground literals in $\mathcal{S}$ to Π:

Definition 4.10 (e-program, $e(\Pi, \mathcal{S})$ [Zhang, 2006])**.** *Given a set of ground literals $\mathcal{S}$, $e(\Pi, \mathcal{S})$ denotes the program obtained from program Π by deleting*

1. *each rule in Π that has a formula $\neg\ell$ in its body with $\ell \in \mathcal{S}$, and*
2. *all formulas of form ℓ in the bodies of the remaining rules with $\ell \in \mathcal{S}$.*

The following example from Zhang illustrates the definition.

Example 4.1 ([Zhang, 2006])**.** *Given $\mathcal{S} = \{a, \sim b\}$ and the program*

$$\begin{aligned}\Pi = \{c &\leftarrow a \\ \sim d &\leftarrow \neg a\}\end{aligned}$$

$e(\Pi, S) = \{c \leftarrow \top\}$.

This definition will prove to be useful to test a *coherence property* in Zhang's approach.

Definition 4.11 (Coherence [Zhang, 2006])**.** *A set of ground literals $\mathcal{S}$ is* coherent *with an extended logic program Π if for any answer set $\mathcal{S}'$ of $e(\Pi, \mathcal{S})$, $\mathcal{S} \cup \mathcal{S}'$ is consistent.*

By continuing Example 4.1, the only answer set of $e(\Pi, S)$ is $\{c\}$. Thus, $\mathcal{S}$ is coherent with Π.

As another example, let us consider Zhang's: $\mathcal{S} = \{a, \sim b\}$ is coherent with $\Pi = \{c \leftarrow a, \sim d \leftarrow \neg a\}$ because the only answer set of $e(\Pi, \mathcal{S})$ is $\{c\}$. However, $\{a, \sim b, \sim c\}$ is not coherent with Π.

These are basic steps towards a general proposal that consists in *two* main steps to perform an update of two programs. Firstly, *eliminating contradictory rules* from a previous program with respect to the last one. Secondly, the semantics solves *conflicts* between the remaining rules of the programs. The semantics that determines the specifications of such an elimination and conflict resolution is *Prioritised Logic Programs*.

4.4.2 Prioritised Logic Programs

In order to specify the algebra for this logic program update proposal, Zhang employs an earlier platform called *Prioritised Logic Programming* [Zhang, 2003b; Zhang and Foo, 1997], or simply PLP. Informally, this sort of logic programs consists of a set of *preference relations* and of a *naming function* that assigns a name to each rule.

Definition 4.12 (Prioritized Logic Program PLP [Zhang, 2006])**.** *A* prioritised logic program $\mathcal{P}$ *is a triple* $(\Pi, \mathcal{N}, <)$*, where* Π *is an* extended logic program*, $\mathcal{N}$ is a* naming function *mapping each rule in* Π *to a name, and "$<$" is a strict* partial order on names*. Moreover, $\mathcal{P}(<)$ denotes the set of* $<$-relations *of* $\mathcal{P}$*.*

According to Zhang, if $\mathcal{N}(\rho) < \mathcal{N}(\rho')$ holds in $\mathcal{P}$, rule ρ is preferred to be applied over rule ρ' when evaluating $\mathcal{P}$. What is "evaluation" of $\mathcal{P}$ anyhow? The following definitions code what an *evaluation of a* $\mathcal{P}$ is. Meanwhile, it is worth recalling from Section 3.2 what an extended logic program is, before going any further:

Definition 4.13 (Extended Logic Program, ELP)**.** *An* extended logic program *is a set of rules of form*

$$\ell \leftarrow \ell_1, \ell_2, \ldots, \ell_m, \neg\ell_{m+1}, \ldots, \neg\ell_n$$

where ℓ_i *is a* literal *and* $0 \leq m \leq n$.

Finally, the definition of a *Defeated rule* looks as follows.

Definition 4.14 (Defeated rule [Zhang, 2003b])**.** *Let* Π *be a ground* extended logic program *and* ρ *an ELP ground rule like in Definition 4.13 —*ρ *does not necessarily belong to* Π. *Rule* ρ *is* defeated by Π *if and only if* Π *has an answer set and for any answer set* $\mathcal{S}$ *of* Π, *there exists some* $\ell_i \in \mathcal{S}$ *where* $m + 1 \leq i \leq n$.

In other words, a program defeats a rule whenever there is a literal in any of the answer sets of the program that is default-negated in the rule.

For example, given a program

$$\begin{aligned} \Pi = \{a &\leftarrow \top \\ c &\leftarrow b \\ d &\leftarrow \neg e\} \end{aligned}$$

whose answer set is $\{a, d\}$, Π defeats rules like

$$\begin{aligned} \bot &\leftarrow \neg d; \\ a &\leftarrow \neg a; \\ c &\leftarrow b, \neg a, \neg d, \neg e \end{aligned}$$

Similarly to the case of extended logic programs, an evaluation of a PLP shall be in terms of its ground form. Moreover, Zhang states that a PLP like $\mathcal{P}' = (\Pi', \mathcal{N}', <')$ is the *ground instantiation* of $\mathcal{P} = (\Pi, \mathcal{N}, <)$ if (1) Π' is the ground instantiation of Π; and (2) $<'$ is a strict partial ordering and $\mathcal{N}'(\rho_1') <' \mathcal{N}'(\rho_2') \in \mathcal{P}'(<')$ if and only if there exist rules ρ_1 and ρ_2 in Π such that ρ_1' and ρ_2' are ground instances of ρ_1 and ρ_2, respectively, and $\mathcal{N}(\rho_1) < \mathcal{N}(\rho_2) \in \mathcal{P}(<)$.

Definition 4.15 ($\mathcal{P}^<$ Reduct [Zhang, 2003b])**.** *Let* $\mathcal{P} = (\Pi, \mathcal{N}, <)$ *be a* prioritised logic program. $\mathcal{P}^<$ *is a* reduct of $\mathcal{P}$ *with respect to "<" if and only if there exists a sequence of sets* Π_i *(*$i = 0, 1, \ldots$*) such that:*

1. $\Pi_0 = \Pi$;

2. $\Pi_i = \Pi_{i-1} \setminus \{\rho_1, \rho_2, \dots\}$ *such that the following two conditions hold:*

   (a) *there exists* $\rho \in \Pi_{i-1}$ *such that for every* j $(j = 1, 2, \dots)$,
   $\mathcal{N}(\rho) < \mathcal{N}(\rho_j) \in \mathcal{P}(<)$
   and $\rho_1, \rho_2, \dots$ *are defeated by* $\Pi_{i-1} \setminus \{\rho_1, \rho_2, \dots\}$

   (b) *there are no rules* $\rho', \rho'', \dots \in \Pi_{i-1}$ *such that*
   $N(\rho_j) < N(\rho'), N(\rho_j) < N(\rho''), \dots$
   for some j $(j = 1, 2, \dots)$ *and* $\rho', \rho'', \dots$ *are defeated by* $\Pi_{i-1} \setminus \{\rho', \rho'', \dots\}$

3. $\mathcal{P}^{<} = \bigcap_{i=0}^{\infty} \Pi_i$.

In Definition 4.15, $\mathcal{P}^{<}$ is an extended logic program that comes from Π by removing some defeated rules from Π, by following the order relations in $\mathcal{P}(<)$ on rules named by $\mathcal{N}$. Specifically, if $\mathcal{N}(\rho) < \mathcal{N}(\rho_1)$, $\mathcal{N}(\rho) < \mathcal{N}(\rho_2)$, ..., and $\Pi_{i-1} \setminus \{\rho_1, \rho_2, \dots\}$ defeats $\{\rho_1, \rho_2, \dots\}$, then, the rules $\rho_1, \rho_2, \dots$ will be out from Π_{i-1} unless a *less preferred rule* than can be removed in turn: conditions (2a) and (2b). One ought to compute the reduct procedure until a *fixed point*. Note that it is said "less preferred" rather than the converse for, at this stage, there is no update semantics.

In addition, condition (2b) in Definition 4.15 is necessary. In its absence, some *counterintuitive results* may be derived —[Zhang, 2006]. For instance, consider the following example from Zhang:

Example 4.2 ([Zhang, 2006])**.** *Consider program*

$\mathcal{P}_1 = (\Pi, \mathcal{N}, <)$*:*
$N_1 : \mathit{flies}(X) \leftarrow \mathit{bird}(X)$, not $\neg \mathit{flies}(X)$
$N_2 : \neg \mathit{flies}(X) \leftarrow \mathit{penguin}(X)$, not $\mathit{flies}(X)$
$N_3 : \mathit{bird}(\mathit{tweety}) \leftarrow \top$
$N_4 : \mathit{penguin}(\mathit{tweety}) \leftarrow \top$
$N_2 < N_1$

If one added the preference $N_3 < N_2$ *in* $\mathcal{P}_1$*, then using a modified version of Definition 4.15 without condition (b), program*

$\{\mathit{flies}(\mathit{tweety}) \leftarrow \mathit{bird}(\mathit{tweety}), \neg\neg \mathit{flies}(\mathit{tweety})$
$\mathit{bird}(\mathit{tweety}) \leftarrow \top$
$\mathit{penguin}(\mathit{tweety}) \leftarrow \top\}$

is a reduct of $\mathcal{P}_1$*, from which it concludes that* Tweety flies*. On the other hand, by considering both the added preference and condition (2b) one will conclude that* Tweety does not fly *from a unique reduct that lacks rule* N_3*.*

Finally, an *interpretation of a PLP* is the answer sets of its reduct, as formally specified in the following definition.

Definition 4.16 (Answer Set of $\mathcal{P}$ [Zhang, 2003b])**.** *Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$ be a* PLP *and Lit the set of all ground literals in the language of $\mathcal{P}$. For any subset $\mathcal{S}$ of Lit, $\mathcal{S}$ is an* answer set of $\mathcal{P}$ *if and only if $\mathcal{S}$ is an answer set for some reduct $\mathcal{P}^<$ of $\mathcal{P}$.*

Using Definition 4.15 and Definition 4.16, it is easy to conclude that $\mathcal{P}_1$ in Example 4.2 has a unique reduct as follows:

$$\begin{aligned}\mathcal{P}_1^< = \{&\neg \mathit{flies}(\mathit{tweety}) \leftarrow \mathit{penguin}(\mathit{tweety}),\ \mathit{not}\ \mathit{flies}(\mathit{tweety})\\ &\mathit{bird}(\mathit{tweety}) \leftarrow \top\\ &\mathit{penguin}(\mathit{tweety}) \leftarrow \top\}\end{aligned}$$

from which one concludes the following answer set of $\mathcal{P}_1^<$:

$$\mathcal{S} = \{\mathit{bird}(\mathit{tweety}),\ \mathit{penguin}(\mathit{tweety}),\ \neg \mathit{flies}(\mathit{tweety})\}.$$

Let us analyse a complete example inspired from the same reference [Zhang, 2006], which illustrates in detail when a PLP has more than one reduct. Before that, one should be aware that a PLP may or may not have answer sets. In the first case, the program is called *well-defined program.*

Example 4.3 ([Zhang, 2006])**.** *Suppose a PLP consisting of*

$$\begin{aligned}\mathcal{P} = (\Pi, \mathcal{N}, >) := \{N_1 &: a \leftarrow \top\\ N_2 &: b \leftarrow \neg c\\ N_3 &: d \leftarrow \top\\ N_4 &: c \leftarrow \neg b\\ &\quad N_1 < N_2, N_3 < N_4\}\end{aligned}$$

By Definition 4.15, one reduct is constructed as

1. $\Pi_0 = \Pi$
2. $\Pi_1 = \Pi_0 \setminus \{b \leftarrow \neg c\}$ *because rule $a \leftarrow \top \in \Pi_0$ and with its tag N_1, one can find the relation $N_1 < N_2 \in \mathcal{P}(<)$ and $b \leftarrow \neg c$ is defeated by $\Pi_0 \setminus \{b \leftarrow \neg c\}$. Last, there are no rules $\rho', \rho'', \cdots \in \Pi_0$, whose tag is "greater than" N_2, and defeated by $\Pi_0 \setminus \{\rho', \rho'', \dots\}$.*
3. *Finally, the reduct is the intersection of the two programs:*

$$\mathcal{P}(<) = \{(a \leftarrow \top); (d \leftarrow \top); (c \leftarrow \neg b)\}$$

and the other reduct as

1. $\Pi_0 = \Pi$
2. $\Pi_1 = \Pi_0 \setminus \{c \leftarrow \neg b\}$ *because rule* $d \leftarrow \top \in \Pi_0$ *and with its tag* N_3 *there is the relation* $N_3 < N_4 \in \mathcal{P}(<)$ *and* $c \leftarrow \neg b$ *is defeated by* $\Pi_0 \setminus \{c \leftarrow \neg b\}$ *and there are no rules* $\rho', \rho'', \cdots \in \Pi_0$*, whose tag is "greater than"* N_4*, and defeated by* $\Pi_0 \setminus \{\rho', \rho'', \dots\}$.
3. *Finally,* $\mathcal{P}(<) = \{(a \leftarrow \top); (b \leftarrow \neg c); (d \leftarrow \top)\}$

This section of prioritised logic programs is the necessary background to give the interpretation of the following procedure for updates under the approach of Zhang, that, as mentioned before, consists in two main steps: *contradiction elimination* and *conflict resolution.*

4.4.3 Eliminating Contradictions

The first step in updating two extended logic programs under Zhang's approach is *eliminating contradictions* by means of an *extended simple-fact update program.* Informally, the extended simple-fact program consist of establishing a high preference to *inertia rules* over *update rules* so that facts in the initial knowledge base may persist after an update. Then, it consists in interpreting the semantics of a resulting (possibly empty) *update program(s)* that should have a *minimal difference with the answer sets* of the original program. This interpretation of the update program(s) is the same as for PLP's.

Before going straight to the main definition, some minor notation is necessary:

Definition 4.17 (Initial Knowledge; PLP Languages [Zhang, 2006])**.** *Let*

$\mathcal{B}$ *denote an initial consistent* knowledge base *of ground literals of a language* $\mathcal{L}$*;*

Π *an update extended logic program over* $\mathcal{L}$*; and*

$\mathcal{L}_{\mathsf{new}}$ *an extension to* $\mathcal{L}$*, by propositional literals of the form* $\mathsf{new}\text{-}\ell \mid \ell \in \mathcal{L}$.

Zhang represents a *generalised simple-fact update* through a triple, which specifies the *changes to an original knowledge base*, according to a new update.

Formally,

Definition 4.18 ($\mathsf{U_{PLP}}$-specification, $\mathsf{U_{PLP}}(\mathcal{B}, \Pi)$ [Zhang, 2006])**.** *Let* $\mathcal{B}$*,* Π*,* $\mathcal{L}$*, and* $\mathcal{L}_{new}$ *be specified as in Definition 4.17. The* specification of updating $\mathcal{B}$ with Π *is a PLP over* $\mathcal{L}_{new}$*, denoted as* $\mathsf{U_{PLP}}(\mathcal{B}, \Pi) = (\Pi^*, \mathcal{N}, <)$*, as follows:*

1. Π^* *consists of following rules:*

Initial knowledge rules: *for each literal* ℓ *in* $\mathcal{B}$*, there is a rule* $\ell \leftarrow \top$

Inertia rules: *for each* predicate symbol[1]

$P \in \mathcal{L}$*, there are two rules:*

$$\textit{new-}P(x) \leftarrow P(x), \neg \sim \textit{new-}P(x)$$

and

$$\sim \textit{new-}P(x) \leftarrow \sim P(x), \neg \textit{new-}P(x)$$

Update rules: *for each rule*

$$\ell_0 \leftarrow \ell_1, ..., \ell_m, \neg \ell_{m+1}, \ldots, \neg \ell_n \in \Pi$$

there is a rule[2]

$$\textit{new-}\ell_0 \leftarrow \textit{new-}\ell_1, \ldots, \textit{new-}\ell_m, \neg \textit{new-}\ell_{m+1}, \ldots, \neg \textit{new-}\ell_n$$

2. Naming function $\mathcal{N}$ *assigns a unique name* N *for each rule in* Π^*.

3. *For any inertia rule* ρ *and update rule* ρ', $\mathcal{N}(\rho) < \mathcal{N}(\rho')$.

According to Zhang, an answer set of Π^* represents *possible resulting knowledge bases* from the update of $\mathcal{B}$ by Π, and a literal new-ℓ represents the *persistence of* ℓ if $\ell \in \mathcal{B}$ or a change of ℓ if $\sim\ell \in \mathcal{B}$ or $\ell \notin \mathcal{B}$ with respect to the update. For instance, in Example 4.4, interpreting Π^* corresponds to simple-fact update semantics, and it has **two answer sets**: $\{\sim a, b, c, newa, newc, \sim newb\}$ and $\{\sim a, b, c, newa, newc, newb\}$, which means that the truth value of b is *indefinite* by $newb$ with respect to the update because the new atom $newb$ is true in one answer set and false in the other, whose conclusion is coming up —(4.9) from Example 4.4.

Up to now, one can transform an initial knowledge base into an initial logic program and *inertial rules* together with the *update rules* to form a PLP. In addition, one can establish *preference relations* among PLP rules, in order to specify a $\mathsf{U}_{\mathsf{PLP}}(\mathcal{B}, \Pi)$. Once a PLP is interpreted, another definition is necessary to get the results of the specifications and to eliminate contradictions of the original sets of rules.

In general, the *interpretations of an update program* U_{PLP} come from the answer sets of its corresponding PLP. Such an interpretation shall lead to one or more possible new knowledge bases, as expressed in the following definition.

[1] A *predicate symbol* corresponds to an *atom*, in my notation.

[2] There is no formal specification in [Zhang, 2006] for strong-negated atoms. However, on the basis of his examples, one may suppose that *for every literal of form* $\sim\ell$ *in the formula, there is a strong-negated form* $\sim$new-ℓ.

Definition 4.19 (Possible Resulting Knowledge Base, $\mathcal{B}$' [Zhang, 2006])**.** *Let* $U_{PLP}(\mathcal{B}, \Pi)$ *be specified as in Definition 4.18. A set* $\mathcal{B}$*' of ground literals is called a* possible resulting knowledge base *with respect to* $U_{PLP}(\mathcal{B},\Pi)$*, if and only if* $\mathcal{B}$*' satisfies the following conditions:*

1. *if* $U_{PLP}(\mathcal{B},\Pi)$ *has a consistent answer set* $\mathcal{S}$*, then* $\mathcal{B}' = \{\ell \mid \textsf{new-}\ell \in \mathcal{S}\}$*;*
2. *if* $U_{PLP}(\mathcal{B},\Pi)$ *does not have a consistent answer set (i.e.,* $U_{PLP}(\mathcal{B},\Pi)$ *is not well defined), then* $\mathcal{B}' = \mathcal{B}$*.*

The name $S_{PLP}(U_{PLP}(\mathcal{B}, \Pi))$ *denotes the set of all* resulting knowledge bases *of* $U_{PLP}(\mathcal{B},\Pi)$*.*

Notice that what Zhang calls *knowledge base* is actually a set of literals.

Now, let us start with a simple but representative and thorough example (proposed by Zhang) that illustrates this process.

Example 4.4 ([Zhang, 2006])**.** *Suppose the initial knowledge base* $\mathcal{B} = \{\sim a, b, c\}$ *and the update program* Π *={(*$\sim b \leftarrow \neg b$*), (*$a \leftarrow c$*)} By Definition 4.18, the corresponding PLP specification is* $U_{PLP}(\mathcal{B}, \Pi) = (\Pi^*, \mathcal{N}, <)$*, consisting of the following rules:*

Initial knowledge: Π_0*:*

$$\sim a \leftarrow \top \qquad b \leftarrow \top \qquad c \leftarrow \top$$

Update rules:

$$u_1: \ \sim newb \leftarrow \neg newb \qquad u_2: \ newa \leftarrow newc$$

Inertia rules:

$$\begin{aligned} i_1: &\ \ newa \leftarrow a, \neg \sim newa \\ i_2: &\ \sim newa \leftarrow \sim a, \neg newa \\ i_3: &\ \ newb \leftarrow b, \neg \sim newb \\ i_4: &\ \sim newb \leftarrow \sim b, \neg newb \\ i_5: &\ \ newc \leftarrow c, \neg \sim newc \\ i_6: &\ \sim newc \leftarrow \sim c, \neg newc \end{aligned}$$

Rule preferences *By Definition 4.18,* $i_i < u_j$ *with* $i, j > 0$ *are*

$$\begin{array}{lll} N(i_1) < N(u_1) & N(i_1) < N(u_2) & N(i_2) < N(u_1) \\ N(i_2) < N(u_2) & N(i_3) < N(u_1) & N(i_3) < N(u_2) \\ N(i_4) < N(u_1) & N(i_4) < N(u_2) & N(i_5) < N(u_1) \\ N(i_5) < N(u_2) & N(i_6) < N(u_1) & N(i_6) < N(u_2) \end{array}$$

With these specifications, one may compute the two answer sets of Π^**. Namely,*

$$\{\sim a, b, c, newa, newc, newb\}$$
$$\{\sim a, b, c, newa, newc, \sim newb\}$$

However, the prioritised logic program $\mathcal{P}$ *has the* ***unique answer set***

$$\{\sim a, b, c, newa, newc, newb\}$$

from its unique reduct because there is only one rule (u_1) *defeated by* $\Pi_0 \backslash \{u_1\}$ *with which one may establish the relations* $N(i_3) < N(u_1) \in \mathcal{P}$ *and there are no less-preferred rules than* u_i *in* $\mathcal{P}(<)$ *—Definition 4.15. As a consequence,* $\mathcal{S}_{PLP}(\mathcal{B}, \Pi) = \{a, b, c\} = \mathcal{S}_{(\Pi_0, \Pi_1)}$ *from Definition 4.19, and the transformed program from* Π_0 *with respect to* Π_1 *that is a maximal subset of* Π_0 *and is coherent with* $\mathcal{S}_{(\Pi_0, \Pi_1)}$ *is just* $\{b, c\}$*. From this program, the* $\mathcal{U}_{\diamond_Z}(\Pi_0, \Pi_1)$ *specification corresponds to the following* $\mathcal{P}$*:*

$$\begin{array}{lll} & \rho_1: & \sim b \leftarrow \neg b \\ & \rho_2: & a \leftarrow c \\ & \rho_3: & b \leftarrow \top \\ & \rho_4: & c \leftarrow \top \\ \rho_1 < \rho_3 & \rho_1 < \rho_4 & \\ \rho_2 < \rho_3 & \rho_2 < \rho_4 & \end{array}$$

whose unique answer set out of the unique reduct ***is*** $\{a, b, c\}$*. That is because defeated rules may not derive from a simple fact update, and that is the difference with the* extended simple fact update.

This *simple-fact update approach* originally appeared in [Marek and Truszczynski, 1994, 1998]. However, it is not adequate for practical applications for the simple reason that, as its name suggests, its definition does not deal with general (non-factual) rules. As a result, Zhang *reformulates the approach to allow updating ELP's, rather than only facts*, by means of the following two definitions, where the first one eliminates contradictory rules between Π_0 and Π_1.

Definition 4.20 (Transformed Program, $\Pi_{(\Pi_0, \Pi_1)}$ [Zhang, 2006])**.** *Given two consistent programs* Π_0 *and* Π_1*, with* $\mathcal{S}_{\Pi_0}$ *as an answer set of* Π_0 *and* $\mathcal{S}_{(\Pi_0, \Pi_1)}$ *as an answer set of the update of* $\mathcal{S}_{\Pi_0}$ *with* Π_1*. Suppose* $\mathcal{S}_{(\Pi_0, \Pi_1)} \in \mathcal{S}_{PLP}(\mathcal{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1))$*. An extended logic program* $\Pi_{(\Pi_0, \Pi_1)}$ *is called a* transformed program *from* Π_0 *with respect to* Π_1*, if*

$\Pi_{(\Pi_0,\Pi_1)}$ *is a maximal subset of the ground instantiation of* Π_0 *such that* $\mathcal{S}_{(\Pi_0,\Pi_1)}$[1] *is coherent with* $\Pi_{(\Pi_0,\Pi_1)}$.

In order to illustrate this definition, see the transformed program in Example 4.5.

Once there is a transformed program, a set of *preferences* between its rules is to solve *possible conflicts*.

Definition 4.21 (Update Specification, $\mathsf{U}_{\diamond\mathsf{Z}}(\Pi_0,\Pi_1)$ [Zhang, 2006])**.** *Let* $\Pi_{(\Pi_0,\Pi_1)}$ *be defined as in Definition 4.20. A* specification of updating Π_0 with Π_1 *is a PLP, denoted as* $U_{\diamond\mathsf{Z}}(\Pi_0,\Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$*, where, for each rule* ρ *in* Π_1 *and each rule* ρ' *in* $\Pi_{(\Pi_0,\Pi_1)}$*, there is a preference relation* $\mathcal{N}(\rho) < \mathcal{N}(\rho')$.

Extended from Zhang, the following example illustrates this definition.

Example 4.5 (Transformed Program)**.** *Consider the following set of programs*

$$\Pi_0 = \{a \leftarrow \top \quad c \leftarrow b \quad d \leftarrow \neg e\} \qquad \Pi_1 = \{b \leftarrow a \quad \sim c \leftarrow b \quad e \leftarrow \neg d\}$$

it is easy to verify that $\mathcal{S}_{\Pi_0} = \{a, d\}$*. Next, its* $U_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1) = (\Pi^*, \mathcal{N}, <)$*, where* Π^* *consists of*

Initial rules:

$$a \leftarrow \top \qquad d \leftarrow \top$$

[1] Note that the original definition must have a typographical error when reading $\mathcal{S}_{\Pi_0}$ rather than $\mathcal{S}_{(\Pi_0,\Pi_1)}$.

Inertial rules:

$$\begin{aligned}
i_1 : newa &\leftarrow a, \neg \sim newa \\
i_2 : \sim newa &\leftarrow \sim a, \neg newa \\
i_3 : newd &\leftarrow d, \neg \sim newd \\
i_4 : \sim newd &\leftarrow \sim d, \neg newd \\
i_5 : newb &\leftarrow b, \neg \sim newb \\
i_6 : \sim newb &\leftarrow \sim b, \neg newb \\
i_9 : newc &\leftarrow c, \neg \sim newc \\
i_{10} : \sim newc &\leftarrow \sim c, \neg newc \\
i_{13} : newe &\leftarrow e, \neg \sim newe \\
i_{14} : \sim newe &\leftarrow \sim e, \neg newe
\end{aligned}$$

Update rules:

$$\begin{aligned}
u_1 : newb &\leftarrow newa \\
u_2 : \sim newc &\leftarrow newb \\
u_3 : newe &\leftarrow \neg newd
\end{aligned}$$

and its $<$*-specifications are*

$$\begin{array}{lll}
N(i_1) < N(u_1); & N(i_1) < N(u_2); & N(i_1) < N(u_3); \\
N(i_2) < N(u_1); & N(i_2) < N(u_2); & N(i_2) < N(u_3); \\
N(i_3) < N(u_1); & N(i_3) < N(u_2); & N(i_3) < N(u_3); \\
N(i_4) < N(u_1); & N(i_4) < N(u_2); & N(i_4) < N(u_3); \\
N(i_5) < N(u_1); & N(i_5) < N(u_2); & N(i_5) < N(u_3); \\
N(i_6) < N(u_1); & N(i_6) < N(u_2); & N(i_6) < N(u_3); \\
N(i_9) < N(u_1); & N(i_9) < N(u_2); & N(i_9) < N(u_3); \\
N(i_{10}) < N(u_1); & N(i_{10}) < N(u_2); & N(i_{10}) < N(u_3); \\
N(i_{13}) < N(u_1); & N(i_{13}) < N(u_2); & N(i_{13}) < N(u_3); \\
N(i_{14}) < N(u_1); & N(i_{14}) < N(u_2); & N(i_{14}) < N(u_3)
\end{array}$$

Note that, as there is no conflict between rules *of inertia and update, the* $<$*-specifications*

of this example do not apply. Next, the unique answer set of Π^* *is just*

$$\{a, d, newa, newd, newb, \sim newc\}$$

which corresponds to the possible resulting knowledge base $\mathcal{S}_{(\Pi_0,\Pi_1)} = \{a, b, \sim c, d\}$*, and* $e(\Pi_0, \mathcal{S}_{(\Pi_0,\Pi_1)}) = \{(a \leftarrow \top), (c \leftarrow \top), (d \leftarrow \neg e)\}$*, whose unique answer set is* $\{a, c, d\}$*. By Definition 4.20, the maximal subset of* Π_0 *that is* coherent *with* $\mathcal{S}_{(\Pi_0,\Pi_1)}$ *is just the unique* ***transformed program*** $\Pi_{(\Pi_0,\Pi_1)} = \{(a \leftarrow \top), (d \leftarrow \neg e)\}$*. Finally, the* ***update specification*** *from* Π_0 *and* Π_1 *is defined as*

$$U_{\diamond z}(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$$

where

$$\begin{aligned} i_1 : &\quad a \leftarrow \top \\ i_2 : &\quad d \leftarrow \neg e \\ t_1 : &\quad b \leftarrow a \\ t_2 : &\quad \sim c \leftarrow b \\ t_3 : &\quad e \leftarrow \neg d \end{aligned}$$

and its $<$***-relations*** *are* $t_x < i_y$*. In addition, the* ***reduct*** *of* $U_{\diamond z}(\Pi_0, \Pi_1)$ *is*

$$\begin{aligned} a &\leftarrow \top \\ b &\leftarrow a \\ \sim c &\leftarrow b \\ e &\leftarrow \neg d \end{aligned}$$

Up to now, a transformed program can eliminate contradictions between an original knowledge base and its update. On the other hand, there are circumstances that do not cause contradiction, but *indefinite conclusions* that must be observed.

4.4.4 Solving Conflicts

In the process of updating a logic program with another, there are rules that might be in conflict when producing indefinite conclusions. The way in which Zhang deals with this problem is by overriding old conflicting rules with the new ones, coded in the preferences of a transformed program, and by producing a called *possible resulting program*, as follows.

Definition 4.22 (Possible Resulting Program, Π_0' [Zhang, 2006])**.** *A program* Π_0' *is a* possible resulting program *of* $U_{\diamond_Z}(\Pi_0, \Pi_1)$ *after updating* Π_0 *with* Π_1 *if* Π_0' *is a reduct of the ground instantiation of* $U_{\diamond_Z}(\Pi_0, \Pi_1)$.

For example, the unique possible resulting program of updating Π_0 with Π_1 is illustrated in Example 4.5.

In summary, this process consists in deriving *possible resulting program*(s) $\Pi_0' = \mathsf{U}_{\diamond_Z}(\Pi_0, \Pi_1)$ —and eventually their answer set(s)— that are the *reduct(s) of a PLP* from an *update specification* $\mathsf{U}_{\diamond_Z}(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$, where $\Pi_{(\Pi_0,\Pi_1)}$ is a *transformed program* of Π_0 and Π_1, derived from a *possible resulting knowledge base* of a *generalised simple-fact update* from the answer sets of the original program and its updating program, which is defined as *initial knowledge*.

A mandatory test is Example 1.1, already introduced in Chapter 1 and recapped in the current chapter as Example 4.3, which produces *counterintuitive results* in many of the existing semantics for updates. So, I will compute it under Zhang's approach as follows.

Observation 4.13. *Suppose an initial program*

$$\begin{aligned} \Pi_0 = \{ day &\leftarrow \neg night \\ night &\leftarrow \neg day \\ stars &\leftarrow night,\ \neg cloudy \\ \sim stars &\leftarrow \top \} \end{aligned}$$

updated with

$$\begin{aligned} \Pi_1 = \{ stars &\leftarrow constls \\ constls &\leftarrow stars \} \end{aligned}$$

Its corresponding PLP specification, $U_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1) = (\Pi^*, \mathcal{N}, <)$*, is as follows:*

Initial Knowledge:

$$\begin{aligned} i_0 :&\quad day \leftarrow \top \\ i_0 :&\quad \sim stars \leftarrow \top \end{aligned}$$

Inertial Rules:

$$
\begin{aligned}
i_1 &: \quad newday \leftarrow day, \neg \sim newday \\
i_2 &: \quad \sim newday \leftarrow \sim day, \neg newday \\
i_3 &: \quad newstars \leftarrow stars, \neg \sim newstars \\
i_4 &: \quad \sim newstars \leftarrow \sim stars, \neg newstars \\
i_5 &: \quad newnight \leftarrow night, \neg \sim newnight \\
i_6 &: \quad \sim newnight \leftarrow \sim night, \neg newnight \\
i_7 &: \quad newconstellations \leftarrow constls, \neg \sim newconstellations \\
i_8 &: \quad \sim newconstellations \leftarrow \sim constls, \neg newconstellations \\
i_9 &: \quad newcloudy \leftarrow cloudy, \neg \sim newcloudy \\
i_{10} &: \quad \sim newcloudy \leftarrow \sim cloudy, \neg newcloudy
\end{aligned}
$$

Update Rules:

$$
\begin{aligned}
u_1 &: newstars \leftarrow newconstellations \\
u_2 &: newconstellations \leftarrow newstars
\end{aligned}
$$

Rule Preferences:

$$
\begin{aligned}
N(i_1) < N(u_1) \qquad & N(i_1) < N(u_2) \\
N(i_2) < N(u_1) \qquad & N(i_2) < N(u_2) \\
\vdots \qquad & \\
N(i_8) < N(u_1) \qquad & N(i_8) < N(u_2)
\end{aligned}
$$

where its unique $\mathcal{S}_{PLP}(\mathcal{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1)) = \{day, \sim stars\}$. *In this case,* $\Pi_{(\Pi_0,\Pi_1)}$ *coincides with* Π_0 *because* $\mathcal{S}_{PLP}(\mathcal{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1))$ *is coherent with* Π_0*—resp.* $\Pi_{(\Pi_0,\Pi_1)}$*, where*

$$
\begin{aligned}
e(\Pi_0, \mathcal{S}_{PLP}(\mathcal{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1))) = \{ day &\leftarrow \neg night \\
stars &\leftarrow night,\ \neg cloudy \\
\sim stars &\leftarrow \top \}
\end{aligned}
$$

and its answer set is $\{day, \sim stars\}$*, which is consistent with* $\mathcal{S}_{PLP}(\mathcal{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1))$*. Thus,* $\Pi_{(\Pi_0,\Pi_1)}$ *is a maximal subset of* Π_0.

Finally, its update specification $U_{\diamond_Z}(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$*, whose possible resulting program is just*

$$\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)} \setminus \{night \leftarrow \neg day\} = \begin{array}{rcl} \{stars & \leftarrow & constls \\ constls & \leftarrow & stars \\ day & \leftarrow & \neg night \\ stars & \leftarrow & night, \neg cloudy \\ \sim stars & \leftarrow & \top\} \end{array}$$

with its expected ***answer set*** $\{day, \sim stars\}$.

Despite this nice behaviour, one of the counter-intuitive examples to Zhang's approach has to do with solving conflicts between rules, where most of the current semantics differ, as first pointed out by Eiter et al. [Eiter et al., 2002]:

Example 4.6. *Suppose an initial knowledge base* $\Pi_0 = \{p \leftarrow \neg q\}$ *being updated with* $\Pi_1 = \{q \leftarrow \neg p\}$*. Its simple-fact update specification corresponds to* $U_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1) = (\Pi^*, \mathcal{N}, <)$*, where*

Initial Knowledge:

$$i_0: \quad p \leftarrow \top$$

Inertial Rules:

$$\begin{array}{ll} i_1: & newp \leftarrow p, \neg \sim newp \\ i_2: & \sim newp \leftarrow \sim p, \neg newp \\ i_3: & newq \leftarrow q, \neg \sim newq \\ i_4: & \sim newq \leftarrow \sim q, \neg newq \end{array}$$

Update Rule

$$u_1: \quad newq \leftarrow \neg newp$$

Preferences

$$N(i_1) < N(u_1) \qquad N(i_2) < N(u_1)$$
$$N(i_3) < N(u_1) \qquad N(i_4) < N(u_1)$$

As a result, its unique answer set $\mathsf{S}_{PLP}(\mathsf{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1)) = \{p\}$ *and none of its* $<$*-relations are used. Next, the minimal subset of* Π_0 *that is coherent with its answer set is just* $\Pi_{(\Pi_0,\Pi_1)} = \Pi_0$*. Then, the update specification of* Π_0 *and* Π_1 *is* $\mathsf{U}_{\diamond_Z}(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$ *where*

$$t: \quad p \leftarrow \neg q$$
$$u: \quad q \leftarrow \neg p$$

and $u < t$*. Finally, its unique reduct* $q \leftarrow \neg p$ *comes from* $(\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}) \setminus \{t\}$ *defeating rule* t*. Therefore, the* ***conclusion*** *of such an update is just* $\{q\}$*.*

Last, besides not satisfying some of the principles already pointed out, one of the major drawbacks of this approach is being limited to only one update to a knowledge base. Namely, it is undefined for update sequences and for *successive updates*, which does not seem to lead to immediate practical use. Although Zhang also suggests an extension to one of his earliest approaches in [Zhang, 2003a] to deal with multiple updates, his proposal still makes the same strong assumptions when deciding between multiple models, as in Example 4.6.

4.5 Logic Approaches

Inspired by the semantics presented in Section 4.1, and similar to the mechanisms suggested in Section 4.3, Osorio and Cuevas; Osorio and Zacarías proposed two approaches to depend upon the logical contents rather than on the syntax of logic programs. Moreover, they provided a case study of new properties out of the intermediate logics characterising ASP, in Section 2.1, by reinterpreting the *AGM-postulates* from Table 2.3 in such a logic, shown later in Table 8.1. Finally, they also proposed a new formulation of Eiter et al.'s operator for one-step updates in a simpler way, without introducing new atoms —see Section 4.1 for further details.

Definition 4.23 ([Osorio and Cuevas, 2007])**.** *Given an update sequence* (Π_1, Π_2) *over a set of atoms* $\mathcal{A}$*, an update program* $\Pi_1 \oplus_2 \Pi_2$ *over* $\mathcal{A}^*$ *consists of the following items:*

(i) all constraints in $\Pi_1 \cup \Pi_2$*;*

(ii) for each rule $\rho \in \Pi_1$
$\ell \leftarrow \mathsf{Body}(\rho), \neg \sim\ell$ *if* $\mathsf{Head}(\rho) = \ell$*;*

(iii) all rules $\rho \in \Pi_2$.

In order to illustrate this process, let us go through the following instance recapped from previous sections.

Example 4.7. *Consider the sequence* (Π_1, Π_2) *from Example 4.1:*

$$\begin{aligned}\Pi_1 \oplus_2 \Pi_2 = \{sleep &\leftarrow night, \neg tvon, \neg \sim sleep\\ watchtv &\leftarrow tvon, \neg \sim watchtv\\ night &\leftarrow \neg \sim night\\ tvon &\leftarrow \neg \sim tvon\\ \sim tvon &\leftarrow pfailure\\ pfailure &\leftarrow \top\}\end{aligned}$$

where the ***answer set*** *of the updated is just* $\{pfailure, sleep, night, \sim tvon\}$*, as one would expect.*

Unfortunately, besides the strong limitation of lacking a method for more than one update, this semantics does not meet the least minimal set of properties proposed in this thesis.

In order to illustrate this claim, let us recap Example 4.3.

Example 4.8. *Consider the initial knowledge base*

$$\begin{aligned}\Pi_1 = \{day &\leftarrow \neg night\\ night &\leftarrow \neg day\\ stars &\leftarrow night, \neg cloudy\\ \sim stars &\leftarrow \top\}\end{aligned}$$

whose unique answer set is $\{day, \sim stars\}$*. Now consider the following update*

$$\begin{aligned}\Pi_2 = \{stars &\leftarrow constls\\ constls &\leftarrow stars\}\end{aligned}$$

A missing feature in Definition 4.23 is how to deal with double strong negation in the fourth rule of Π_1*. Taking advantage of* $\mathcal{N}_2$ *logic, one of its axioms gives the equivalence: double negations yields a positive atom. Accordingly, the* $\oplus_2$ *-updated program is*

$$
\begin{aligned}
day &\leftarrow \neg night, \neg \sim day \\
night &\leftarrow \neg day, \neg \sim night \\
stars &\leftarrow night, \neg cloudy, \neg \sim stars \\
\sim stars &\leftarrow \neg stars \\
stars &\leftarrow constls \\
constls &\leftarrow stars
\end{aligned}
$$

As a result, some **extra answer sets** make $\oplus_2$ -operator fail to comply with postulate $(\mathsf{R}\dot{+}4)$ and SC property: $\{day, \sim stars\}$; $\{night, \sim stars\}$; $\{night, stars, constls\}$. Operator $\oplus_3$ is also limited to one update and lacks some other properties too, as summarised in Table A. On the other hand, their corresponding $\oplus_1$ -operator is $\lhd$-operator in Section 4.1 for the case of single updates. Finally, $\oplus_1 \equiv_{\mathsf{ASP}} \oplus_3$ as they themselves prove it, which helps complete the column of $\lhd$-operator in table A.

4.6 Conclusions for Chapter 4

This chapter is a survey of update semantics in ASP in a wide range from *simple-fact one-step updates* of logic programs, up to unlimited updates in a sequence. Some of these works present a vast collection of postulates and principles, and/or implementation. However, all the proposals here introduced still present drawbacks either for being limited to only one update, or for relying on *syntactical principles* to change the original logic program that leads to *counterintuitive results*. In need of a *general semantics* that meets as many principles as possible, upcoming chapters shall present proposals to overcome the mentioned limitations and to meet general principles. In the next chapter, the reader may find a summary of representative examples presented to a certain extent, as well as a summary of properties in Appendix A.

On the other hand, for practical and comercial reasons all of the proposals here presented are called semantics for *updates*, although there are implicit operations in some of them that may suggest a different classification, say *semantics for belief revisions*. The difference is a technical issue, as briefly introduced in Section 2.4, and shall be recapped in upcoming chapters.

Chapter 5

Observations and Examples

This chapter is an arrangement of specific *key observations*, **already introduced in previous sections**, along with a discussion of what they intend to show in general. They are examples of updates and are classified according to general concepts of *vacuous information*, *object-level updates*, *conflicting information* and *initialisation*. Besides the examples and their descriptions, each section shows a summarised comparison of the results when evaluated in each of the four semantics from Chapter 4.

5.1 Vacuous Information

Vacuous or *inert information* shall be an intuitive term in this thesis to describe knowledge that may be an extension to an initial one, but should not produce major changes after its update. In particular, the main motivation to this dissertation is the following example from Section 1:

Observation 5.1. *Suppose an agent who believes that* when *it is* day *it is not* night *and vice versa, and that there are* stars *when it is* night *and when there are no* clouds. *Finally, that at the current moment it is a fact that there are no* stars. *This simple story may be coded*[1] *into* Π_1 *as follows:*

$$\begin{aligned} \Pi_1 = \{ day &\leftarrow \neg night \\ night &\leftarrow \neg day \\ stars &\leftarrow night, \neg cloudy \\ \sim stars &\leftarrow \top \} \end{aligned}$$

whose unique answer set is $\{day, \sim stars\}$. *Later, the agent acquires new information*

[1] Notice that there are other ways to represent the story. The problem is, however, what to do in this particular situation, when the agent runs across this piece of information.

stating that stars *and* constls *(*constellations*) are the same thing, as coded in* Π_2*. As soon as the agent updates* Π_1 *with program*

$$\begin{aligned}\Pi_2 = \{stars &\leftarrow constls\\ constls &\leftarrow stars\}\end{aligned}$$

the expanded alphabet *of the two programs contains only one new* extra atom *with respect to* Π_1*: constls. As the model of* Π_2 *is obviously the empty answer set, constls is considered* synonym *of stars by means of* Π_2*, and thus the update should not change the* original beliefs*. However, the update yields an* extra answer set *in some of the existing update semantics based on the* causal rejection principle *—Section 4.2:*

$$\{stars, constls, night\}$$

which does not coincide with common intuition*.*

The reason is that, although stars can not be true, introducing constls gives another possibility for stars to be true. Thus, the additional answer set is implied.

In general, these supplementary rules in the update are a conservative extension *[Osorio et al., 2001] to* Π_1*: the original language is extended and all answer sets ought to be extensions of the old answer sets. In this specific situation, constls should be true if and only if stars is true.*

This example shows an expansion to the original alphabet of a knowledge base, known in the literature as *conservative extension* of such alphabet. In this case, the rules in Π_2 mean that *constls* is a synonym of *stars*. As a result, one would expect that the original beliefs before the update do not change except that constellations have exactly the same truth value than stars.

In the literature of updates of logic programs, however, various of the most relevant semantics give counterintuitive results with this example, as shown in the following summary.

Eiter's et alii —Section 4.1: Observation 4.3 shows that the corresponding semantics results in counterintuitive update answer sets, $\{day, \sim stars\}$ as well as $\{night, stars, constls\}$.

DyLP—Section 4.2: As shown in Observation 4.5, the resulting dynamic stable models is $\{day\}$ without $\{night, stars, constls\}$.

Sakama-Inoue's —Section 4.3: This semantics computes $\{day, \sim stars\}$ only, as one would expect.

Zhang's —Section 4.4: The unique answer set in this semantics, $\{day, \sim stars\}$, as shown in Observation 4.13, coincides with common intuition.

As a result, the only semantics that does not meet the expected model is Eiter's et alii from Section 4.1, for its base on the causal-rejection principle. On the other hand, DyLP does meet the expected result when using its refined version.

The following observation illustrates a variant of updates that should produce no extra models, first introduced in Observation 4.7.

Observation 5.2. *Suppose an initial knowledge base* $\Pi_0 = \{(c \leftarrow r), (r \leftarrow \top)\}$ *updated with* $\Pi_1 = \{\text{not}\, r \leftarrow \text{not}\, c\}$. *Firstly, the initial generalised stable* ***model*** *of* Π_0 *is* $\{c, r\}$, *and one would expect no changes after the update. However, the update* $\mathcal{P} = \Pi_0 \oplus_R \Pi_1$ *has the* ***extra model*** $\mathcal{M} = \{not\_c, not\_r\}$ *because* $\overline{\mathcal{M}} = \{not\_c, not\_r\}$; $\mathit{Rej}(\mathcal{P}, \mathcal{M}) = \{r \leftarrow \top\}$; $\mathit{Def}(\mathcal{P}, \mathcal{M}) = \{not\_c\}$; *and* $\mathit{least}[(\Pi_0 \cup \Pi_1) \setminus \mathit{Rej}(\mathcal{P}, \mathcal{M}) \cup \mathit{Def}(\mathcal{P}, \mathcal{M})] = \{not\_c, not\_r\} = \overline{\mathcal{M}}$.

The purpose of this example, inspired by Eiter et al., is to show that making knowledge more precise may produce counterintuitive models in some semantics, where one would expect no extra models. As a result, the rest of the semantics analysed in previous chapters show the corresponding behaviour summarised as follows.

Eiter's et alii —Section 4.1: The semantics gives the update answer set $\{c, r\}$ and an extra $\{\sim r\}$.

DyLP—Section 4.2: The resulting dynamic stable models are also both $\{c, r\}$ and the extra $\{not\_c, not\_r\}$, as shown above in Observation 5.2.

Sakama-Inoue's —Section 4.3: This semantics computes just $\{c, r\}$, as one would expect.

Zhang's —Section 4.4: This semantics computes just $\{c, r\}$.

To sum up, both Eiter's et alii (Section 4.1) and DyLP (Section 4.2) show counterintuitive behaviour when updating an initial knowledge base with *vacuous information.*

5.2 Updates at the Object Level

The following example shows how to perform updates at the object level, rather than the meta-level, introduced in Observation 4.8.

Observation 5.3. *Suppose an update to the knowledge base*

$$\begin{aligned} \Pi_1 = \{ \mathit{sleep} &\leftarrow \neg \mathit{tvon} \\ \mathit{watchtv} &\leftarrow \mathit{tvon} \\ \mathit{tvon} &\leftarrow \top \} \end{aligned}$$

with[1]

$$\Pi_2 = \{ pfailure \leftarrow \top \quad \bot \leftarrow pfailure, tvon \}$$

The situation is to be coded into the extended abductive program $\langle\, \Pi_1 \cup \Pi_2, \Pi_1 \setminus \Pi_2 \,\rangle$. *The update program* $\mathcal{UP}$ *of* $\langle\, (\Pi_1 \cup \Pi_2)^{\mathrm{n}}, (\Pi_1 \setminus \Pi_2)^{\mathrm{n}} \,\rangle$ *is specified as*

$$\begin{aligned} \mathcal{UP}: \quad & pfailure \leftarrow \top \\ & \bot \leftarrow pfailure, tvon \\ & sleep \leftarrow \neg tvon, \gamma_1 \\ & watchtv \leftarrow tvon, \gamma_2 \\ & abd(tvon), abd(\gamma_1), abd(\gamma_2), \\ & tvon^{-} \leftarrow \neg tvon \\ & \gamma_1^{-} \leftarrow \neg\gamma_1 \\ & \gamma_2^{-} \leftarrow \neg\gamma_2 \end{aligned}$$

where γ_1 *and* γ_2 *are names of the abducible rules in* $\Pi_1 \setminus \Pi_2$. *Then,* $\mathcal{UP}$ *has the unique U-MAS*

$$\{ pfailure, sleep, \overline{tvon}, tvon^{-}, \gamma_1, \gamma_2 \}$$

which represents the deletion of fact tvon from $\Pi_1 \cup \Pi_2$. *Accordingly, the update of* Π_2 *to* Π_1 *is the resulting program*

$$\begin{aligned} \Pi_3: \quad & sleep \leftarrow \neg tvon \\ & watchtv \leftarrow tvon \\ & pfailure \leftarrow \top \\ & \bot \leftarrow pfailure, tvon \end{aligned}$$

whose ***answer set*** *is just* $\{pfailure, sleep\}$

Next, suppose ***yet another update*** $(\Pi_4 : \neg pfailure \leftarrow \top)$ *to* Π_3, *which represents that there is power again. Sakama and Inoue code this new pair by the abductive program*

[1] In [Alferes et al., 1999] the rule "$\leftarrow pfailure, tvon$" is given as "$\neg tvon \leftarrow pfailure$". These two rules are semantically equivalent under the answer set semantics, as the authors explain in [Sakama and Inoue, 2003]. However, as later seen in this example, the difference between either expression would result in the existence of $\sim tvon$ in the corresponding model!

$\langle\, \Pi_3 \cup \Pi_4, \Pi_3 \setminus \Pi_4 \,\rangle$. *So, the update program of* $\langle\, (\Pi_3 \cup \Pi_4)^{\mathrm{n}}, (\Pi_3 \setminus \Pi_4)^{\mathrm{n}} \,\rangle$ *turns into*

$$\begin{aligned}
\mathcal{UP}: \quad & \neg pfailure \leftarrow \top \\
& sleep \leftarrow \neg tvon\gamma_1 \\
& watchtv \leftarrow tvon\gamma_2 \\
& \bot \leftarrow pfailure, tvon\gamma_3 \\
& abd(pfailure), abd(\gamma_1), abd(\gamma_2), abd(\gamma_3), \\
& pfailure^{-} \leftarrow \neg pfailure \\
& \sim\!\gamma_1 \leftarrow \neg\gamma_1, \sim\!\gamma_2 \leftarrow \neg\gamma_2, \sim\!\gamma_3 \leftarrow \neg\gamma_3.
\end{aligned}$$

Then, $\mathcal{UP}$ *has the unique* U-MAS

$$\{\neg pfailure, sleep, \gamma_1, \gamma_2, \gamma_3, \overline{pfailure}, \sim\!pfailure\},$$

which implies that the result of the update is $(\Pi_3 \cup \Pi_4) \setminus \{pfailure \leftarrow \top\}$. *As a result, the* ***unique answer set*** *of the unique resulting program is* $\{\sim\!pfailure, sleep\}$.

The purpose of Observation 5.3 is to illustrate that various semantics produce counterintuitive results when changing the truth value of a belief with no intuitive justification. In particular, one would expect that, after the second update, the belief set $\{pfailure, sleep\}$ changes to $\{\sim\!pfailure, sleep\}$. The following summary is an outlook of the behaviour of major semantics for updates in the literature.

Eiter's et alii —Section 4.1: Observation 4.2 shows that the corresponding semantics gives the counterintuitive update answer set, $\{tvon, watchtv, \sim\!pfailure\}$.

DyLP—Section 4.2: The corresponding resulting dynamic stable model, as shown in Observation 4.6, is $\{tvon, watchtv\}$, which is clearly counterintuitive .

Sakama-Inoue's —Section 4.3: This semantics computes

$\{\sim\!pfailure, sleep\}$, as one would expect.

Zhang's —Section 4.4: This semantics is unable to perform more than one update.

In conclusion, the only semantics to meet the expected result is Sakama-Inoue's — Section 4.3. The others' problem is either a meta-level update or a lack of a definition to perform multiple updates.

5.3 Conflicting Information

The following observation illustrates how to solve conflicting updates that are not necessary contradictory in their semantics, introduced in Observation 4.12.

Observation 5.4 ([Zhang and Foo, 2005])**.** *Suppose*

$$\begin{aligned}\Pi_0 = \{member(a,g) &\leftarrow \top\\ member(b,g) &\leftarrow \top\\ access(a,f_2) &\leftarrow \top\\ access(X,f_1) &\leftarrow member(X,g)\\ \sim access(X,f_2) &\leftarrow member(X,g), \neg access(X,f_2)\}\end{aligned}$$

updated with

$$\begin{aligned}\Pi_1 = \{member(c,g) &\leftarrow \top\\ \sim access(X,f_1) &\leftarrow member(X,g)\\ access(X,f_2) &\leftarrow member(X,g), \neg \sim access(X,f_2)\}\end{aligned}$$

According to Zhang, this update ought to have the ***unique answer set***

$$\begin{aligned}S = \{&\sim access(a,f_1), \sim access(b,f_1), \sim access(c,f_1),\\ &access(a,f_2), access(b,f_2), access(c,f_2)\}\end{aligned}$$

The purpose of Observation 5.4 is to show how to solve conflicting information that derives indefinite models, as the rest of the semantics, that present four models of the grounded programs equivalent to the following:

$$\begin{aligned}\{&member(c,g), \sim access(c,f_1), member(a,g), member(b,g), access(a,f_2),\\ &\sim access(c,f_2), \sim access(b,f_2), \sim access(a,f_1), \sim access(b,f_1)\}\\ \{&member(c,g), \sim access(c,f_1), member(a,g), member(b,g), access(a,f_2),\\ &\sim access(c,f_2), access(b,f_2), \sim access(a,f_1), \sim access(b,f_1)\}\\ \{&member(c,g), \sim access(c,f_1), member(a,g), member(b,g), access(a,f_2),\\ &access(c,f_2), \sim access(b,f_2), \sim access(a,f_1), \sim access(b,f_1)\}\\ \{&member(c,g), \sim access(c,f_1), member(a,g), member(b,g), access(a,f_2),\\ &access(c,f_2), access(b,f_2), \sim access(a,f_1), \sim access(b,f_1)\}\end{aligned}$$

In order to simplify matters, the problem may be analogous to a simpler and more representative one, introduced earlier in Example 4.6, Section 4.4.4:

Observation 5.5. *Suppose an initial knowledge base* $\Pi_0 = \{p \leftarrow \neg q\}$ *being updated with* $\Pi_1 = \{q \leftarrow \neg p\}$*. Its simple-fact update specification corresponds to* $U_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1) = (\Pi^*, \mathcal{N}, <)$*, where*

Initial Knowledge:

$$i_0: \quad p \leftarrow \top$$

Inertial Rules:

$$\begin{aligned} i_1: &\quad newp \leftarrow p, \neg \sim newp \\ i_2: &\quad \sim newp \leftarrow \sim p, \neg newp \\ i_3: &\quad newq \leftarrow q, \neg \sim newq \\ i_4: &\quad \sim newq \leftarrow \sim q, \neg newq \end{aligned}$$

Update Rule

$$u_1: \quad newq \leftarrow \neg newp$$

Preferences

$$\begin{aligned} N(i_1) < N(u_1) \qquad & N(i_2) < N(u_1) \\ N(i_3) < N(u_1) \qquad & N(i_4) < N(u_1) \end{aligned}$$

As a result, its unique answer set $\mathsf{S}_{PLP}(\mathsf{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1)) = \{p\}$ *and none of its* $<$*-relations are used. Next, the minimal subset of* Π_0 *that is coherent with its answer set is just* $\Pi_{(\Pi_0,\Pi_1)} = \Pi_0$*. Then, the update specification of* Π_0 *and* Π_1 *is* $\mathsf{U}_{\diamond_Z}(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$ *where*

$$\begin{aligned} t: &\quad p \leftarrow \neg q \\ u: &\quad q \leftarrow \neg p \end{aligned}$$

and $u < t$*. Finally, its unique reduct* $q \leftarrow \neg p$ *comes from* $(\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}) \setminus \{t\}$ *defeating rule* t*. Therefore, the* ***conclusion*** *of such an update is just* $\{q\}$*.*

The purpose of this example, due to Eiter et al., is to show that solving conflicts in the way Zhang does has an impact in very specific applications but might not be suitable for a general semantics for updates of logic programs. As a result, the rest of the semantics analysed in previous chapters show the behaviour summarised as follows.

Eiter's et alii —Section 4.1: The semantics gives the update answer sets $\{p\}$ and

$\{q\}$, as one would expect.

DyLP—Section 4.2: The corresponding resulting dynamic stable models are both $\{p\}$ and $\{q\}$.

Sakama-Inoue's —Section 4.3: This semantics computes $\{p\}$ and $\{q\}$, as one would expect.

Zhang's —Section 4.4: This semantics computes just $\{q\}$.

To sum up, Zhang's (Section 4.4) is the only semantics that fails to meet the expected models, due to its formulation to behave just like that: a current rule overrides an earlier one when both of them together produce indefinite results. In very particular application, however, that behaviour may be an advantage, as they suggest in Observation 5.4.

5.4 Initialisation

The following observation illustrates updates to an empty knowledge base, first introduced in Observation 5.6.

Observation 5.6. *Suppose the initial knowledge base* $\Pi = \emptyset$ *updated by a simple fact* $\Pi_1 = \{x \leftarrow \top\}$. *Following Sakama and Inoue's framework, the answer sets of its update program is empty:* $\mathcal{UP} = (\emptyset \setminus \{x \leftarrow \top\}) \cup \mathcal{UR}$, *where* $\mathcal{UR}$ *is clearly empty because the extended abductive program from the update pair has no abducibles:* $\mathcal{A}^* = \emptyset \setminus \{x \leftarrow \top\}$.

Eiter's et alii —Section 4.1: Observation 4.2 shows that the corresponding semantics gives the update answer set, $\{x\}$, as one would expect.

DyLP—Section 4.2: The corresponding resulting refined dynamic stable model is $\{x\}$ as well.

Sakama-Inoue's —Section 4.3: Surprisingly, this semantics computes $\{\}$, which is clearly counterintuitive.

Zhang's —Section 4.4: $\{x\}$ is a model of his semantics.

To conclude this section, the only semantics to fail this property is Sakama-Inoue's from Section 4.3, for its strong syntactic-approach.

5.5 Conclusions for Chapter 5

The purpose of this chapter has been to classify the semantics presented in Chapter 4 according to a general type of problem they exhibit, by means of *key examples* presented up to Chapter 4. Each key example has a description of its purpose as well

as a comparison of its results when computed by every semantics presented in Chapter 4. Accordingly, the empirical results of this survey show that each of the analysed semantics have advantages and disadvantages for the specific purpose they were formulated. In consequence, there is no known general semantics that can cope with all the key-problems in this chapter.

Chapter 6

Relaxing Knowledge-bases

One of the traditional and general goals of belief updates is dealing with new information that might contradict current knowledge. As shown in the survey of Chapter 4, a number of researchers have put forward approaches to update knowledge coded in ASP. However, there are particular rare challenging (even so possible) situations that might lead to *counterintuitive models* of the environment that have been subject of recent research and matter of formulation of new principles. Some of the latest proposals to overcome such missing properties have appeared in [Alferes et al., 2005; Guadarrama, 2008c; Osorio and Zacarías, 2003, 2004] and the properties, amongst others, are compiled in a literature-review unit, in Chapter 2 and summarised in Appendix A.

In particular, although most of those proposals have a strong ASP foundation, some of them are highly dependent on the *syntax* of the programs, which leads to counterintuitive results. In general, they do not observe the basic set of structural properties for updates discussed in Chapter 4, although their original goals are to provide an *update characterisation* to their particular proposes.

Having said that, this chapter consists of a set of *properties to be met* and a basic alternative method to overcome those deficiencies in current approaches mentioned in Chapter 4. The general feature in this proposal is to depend upon the *logical contents* of programs rather than on their syntax, based on a formalism introduced in Section 3.6: *Generalised Answer Sets* by Kakas and Mancarella and employed in [Balduccini and Gelfond, 2003]. Such a formalism shall prove to be a foundation to characterise a basic method of *relaxation* by two main structural properties: *Weak Irrelevance of Syntax* and *Strong Consistency*. Other relevant properties are also a basis to an extended particular formulation in Chapter 7. Additionally, Section 6.3 includes an alternative to compute *Minimal Generalised Updates* by Preferred Answer Sets on *Ordered-disjunctive Logic Programming*, ODLP in [Brewka, 2002; Brewka et al., 2002], by means of a translation.

Last but not least, it good to recap that, for historical and practical reasons, we call these frameworks *update semantics* without an explicit *difference between belief revision and updates*. The technical difference has been introduced in Section 2.4 and shall be made explicit in upcoming chapters. Meanwhile, let us analyse the problem of syntax

dependency in the next sections.

6.1 Model Choice

Over the last years several approaches have been formulated to update logic programs in Answer Set semantics [Alferes et al., 2005, 1999; Eiter et al., 2000a, 2005; Osorio and Zacarías, 2003, 2004; Sakama and Inoue, 1999, 2003; Zhang, 2006; Zhang and Foo, 2005]. As already explined in Chapter 4, most of these works are based upon particular notions of a *(refined) causal rejection of rules* by [see Alferes et al., 2005, 1999], which enforces that, in case of conflicts between rules, *more recent rules are preferred and older rules are overridden.* Such a principle has been particularly studied by [Eiter et al., 2000a,b, 2001, 2002] and reformulated by Alferes et al.[1], although its new formulation still remains limited —Section 4.2. Other approaches are not that explicit in using a syntactical approach but they do follow it, as introduced in Chapter 4.

Inspired by the studies of the above citations and in particular from [Osorio and Zacarías, 2004], this section comprises an alternative solution to the latter, as an application of a formalism called *Minimal Generalised Answer Sets*, MGAS, introduced in Section 3.6.

It is worth noting that this chapter just considers *update pairs* rather than *update sequences*, as an introduction of the proposed properties and of a *method to depend upon models.* Notwithstanding, this chapter shows the basic formulation to relax knowledge bases and to compute them with ODLP. It also leads to an extension, in Chapter 7, as a study to update sequences of knowledge bases.

Formally, an *update pair* is a tuple (Π_1, Π_2) of logic programs over $\mathcal{A}$ if and only if $\mathcal{A}$ represents the set of atoms occurring in $\Pi_1 \cup \Pi_2$.

As a consequence, an update program corresponds to an *abductive program* from the pair, as expressed in the following statement.

Definition 6.1 ($\oslash$-Update Program)**.** *Given an update pair* $\mathbf{\Pi} = (\Pi_1, \Pi_2)$ *of extended logic programs over a set of atoms* $\mathcal{A}$*, an* update program $\mathbf{\Pi}_{\oslash} = \Pi_1 \oslash \Pi_2$ *corresponds to the abductive program* $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$*, where* $\mathcal{A}^*$ *extends* $\mathcal{A}$ *by new unique abductive atoms and* Π' *is constructed as follows:*

(i) *all constraints in* Π_1*.*

(ii) *for each non-constraint rule* $\rho \in \Pi_1$ *there is a unique abducible* α *(a new atom) and the rule is replaced by* $\mathit{Head}(\rho) \leftarrow \mathit{Body}(\rho), \neg\alpha$*.*

[1] Note that [Alferes et al., 2005] uses a refined principle of rejection of rules to overcome *some* of the drawbacks pointed out in this work, but they have to make particular transformations in order to be classified in Answer Set Programming. Unfortunately, the ASP does not seem to be reflected in the class of updating programs used at the front end, for their ultimate goal seems to be Well Founded Semantics.

where $\oslash$ *represents the the corresponding* update operator.

In order to illustrate this method, see Π' in Example 6.1.
Next, the following definition states the interpretation of such an update program.

Definition 6.2 ($\oslash$-update Answer Set)**.** *Let* $\mathbf{\Pi} = (\Pi_1, \Pi_2)$ *be an update pair over a set of atoms* $\mathcal{A}$*. Then,* $\mathcal{S} \subseteq \mathcal{A}$ *is an* update answer set *of* $\mathbf{\Pi}$ *if and only if* $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ *for some* minimal generalised answer set $\mathcal{S}'$ *of* $\mathbf{\Pi}$*.*

The following example illustrates a daily update regarding energy flaw, and it is an adaptation from the original examples in [Alferes et al., 1999] and [Eiter et al., 2002].

Example 6.1.

$$\begin{aligned}
\Pi_1 = \{sleep &\leftarrow \neg tv(on)\\
night &\leftarrow \top\\
watch(tv) &\leftarrow tv(on)\\
tv(on) &\leftarrow \top\}\\
\Pi_2 = \{\sim tv(on) &\leftarrow power(failure)\\
power(failure) &\leftarrow \top\}
\end{aligned}$$

By following Observation 4.1, the single answer set of $\Pi_1 \triangleleft \Pi_2$ (under [Eiter et al., 2002] approach) is just **as one would expect**:

$$\{power(failure), \sim tv(on), sleep, night\} \tag{6.1}$$

On the other hand, by codifying this example under $\oslash$ operator, Π_1 is transformed as follows: for each rule in Π_1, there is a new atom from the set of abducibles $\mathcal{A}^*$. Next, each abducible ought to be default-negated and appended to the body of every rule in Π_1. As a consequence, the update program is the abductive program $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$, where $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ and

$$\begin{aligned}
\Pi' \cup \Pi_2 = \{sleep &\leftarrow \neg tv(on), \neg \alpha_1\\
night &\leftarrow \neg \alpha_2\\
watch(tv) &\leftarrow tv(on), \neg \alpha_3\\
tv(on) &\leftarrow \neg \alpha_4\\
\sim tv(on) &\leftarrow power(failure)\\
power(failure) &\leftarrow \top\}
\end{aligned}$$

whose unique update answer set, out of the unique MGAS

$$\{night, sleep, power(failure), \sim tv(on)\}_{\{\sim\alpha_1, \sim\alpha_2, \sim\alpha_3, \alpha_4\}}$$

coincides with (6.1).

As Example 1.1 shows in Chapter 1, there are some particular cases that most current semantics solve in a *counterintuitive* way, like introducing extra models where one would not expect[1]. However, the operator presented in this chapter can already cope with them.

Example 6.2 (continued)**.** *Suppose the same update pair as in Example 1.1. Under the approach just presented in this chapter,* $\Pi_1 \oslash \Pi_2$ *is the abductive program transformation* $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$ *where* $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ *and*

$$\begin{aligned} \Pi' \cup \Pi_2 = \{ day &\leftarrow \neg night, \neg \alpha_1 \\ night &\leftarrow \neg day, \neg \alpha_2 \\ see(stars) &\leftarrow night, \neg cloudy, \neg \alpha_3 \\ \sim see(stars) &\leftarrow \neg \alpha_4 \\ stars &\leftarrow constellations \\ constellations &\leftarrow stars \} \end{aligned}$$

The MGAS *of this program is* $\{\sim stars, day\}_{\{\sim\alpha_1, \sim\alpha_2, \sim\alpha_3, \sim\alpha_4, \sim\alpha_5, \sim\alpha_6\}}$ *and the sole* ***update answer set****:* $\{\sim stars, day\}$ *that coincides with our intuition.*

After having seen a couple of examples, the following section consists of a set of generalised fundamental properties properties. Many more examples may be tested automatically via online at `http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html`.

6.2 Structural Properties for Updates in ASP

As above mentioned, this chapter is a proposal of just a small set of fundamental properties an update semantics ought to meet, but they should serve as an initial basis of further extensions, like in Chapter 7.

An appropriate complete motivation for this section has already been introduced in Chapter 4 and this set of properties highlights the main *difference with other proposals*, in particular with the ones in such chapter.

The main contribution of this chapter is a method to update knowledge bases depending upon the *semantical contents* rather than syntactical approaches to discard conflicting clauses. Such a goal may be coded into a set of *structural properties* hereby introduced and characterising $\oslash$-operator to perform updates and to overcome the problems discussed in Chapter 4.

[1] See Chapter 4 for further discussions.

⊘-SP-2, Initialisation [Eiter et al., 2002]: $\emptyset \oslash \Pi \equiv \Pi$.

This property states that the update of an initial empty knowledge base yields just the update itself.

⊘-SP-3, Inertia: If Π is consistent, $\Pi \oslash \emptyset \equiv \Pi$.

A consistent theory is in effect unless new *evidence* states otherwise.

⊘-SP-4, Idempotence [Eiter et al., 2002]: $\Pi \oslash \Pi \equiv \Pi$.

This property means that the update of program Π with itself has no effect.

⊘-SP-6, Non-interference, WNI: [Eiter et al., 2002]: If Π_1 and Π_2 are programs defined over *disjoint alphabets*, and either both of them have answer sets or not, then $\Pi_1 \oslash \Pi_2 \equiv \Pi_2 \oslash \Pi_1$.

This property is a specialisation from Eiter et al.'s and implies that the *order of updates* that do not interfere with each other, does not matter.

⊘-SP-7, Augmented Update [Eiter et al., 2002]: If $\Pi_1 \subseteq \Pi_2$ then $\Pi_1 \oslash \Pi_2 \equiv \Pi_2$.

Updating with additional rules makes the previous update obsolete.

⊘-SP-8, Strong Consistency, SC: If $\Pi_1 \cup \Pi_2$ is *consistent*, then $\Pi_1 \oslash \Pi_2 \equiv \Pi_1 \cup \Pi_2$.

The update coincides with the union when $\Pi_1 \cup \Pi_2$ is consistent. This property corresponds to *Bordiga's principle*, studied by [Katsuno and Mendelzon, 1991b].

⊘-SP-9, Weak Irrelevance of Syntax, WIS: Let Π, Π_1, and Π_2 be logic programs under the same language. If $T_{\mathcal{N}_2}(\Pi_1) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_2)$ then $\Pi \oslash \Pi_1 \equiv \Pi \oslash \Pi_2$.

It means that if we update a program Π with Π_1 or with Π_2, the result should depend upon the *logical contents* of Π_1 and Π_2, rather than the particular syntax to spell them.

This property corresponds to *Dalal*'s *Principle of Irrelevance of Syntax*, studied in [Katsuno and Mendelzon, 1991b], and it says that if one updates a program Π with Π_1 (or Π_2), the result should depend upon the *logical contents* of Π_1 (or Π_2), and not on the particular syntax employed to write Π_1 (or Π_2).

The following result is a formal contribution of this chapter.

Theorem 6.1. *⊘-operator satisfies Structural Properties ⊘-SP-2 to ⊘-SP-9.*

Proof. **⊘-SP-2, Initialisation:** $\emptyset \oslash \Pi \equiv \Pi$.

$\emptyset \oslash \Pi$ has the update program $\langle \emptyset \cup \Pi, \emptyset \rangle$, whose MGAS's $\mathcal{M}_\emptyset$ correspond to the answer sets $\mathcal{M}$ of Π. Hence, $\emptyset \oslash \Pi \equiv \Pi$.

⊘-SP-3, Inertia: If Π is consistent, $\Pi \oslash \emptyset \equiv \Pi$.

$\Pi \oslash \emptyset$ has the update program $\langle \Pi' \cup \emptyset, \emptyset \rangle$, whose MGAS $\mathcal{M}_\emptyset$ correspond to the answer sets $\mathcal{M}$ of Π'. Therefore, $\Pi \oslash \emptyset \equiv \Pi$.

⊘-SP-8, Strong Consistency: If $\Pi_1 \cup \Pi_2$ has at least one answer set, then $\Pi_1 \oslash \Pi_2 \equiv \Pi_1 \cup \Pi_2$.

Assume $\mathcal{M}$ is an answer set of $\Pi_1 \cup \Pi_2$. Then, $\mathcal{M}$ is the same model of $\Pi_1 \oslash \Pi_2$. As $\Pi_1 \oslash \Pi_2 = \langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$, and an answer set of $\Pi' \cup \Pi_2 \cup \{H \leftarrow \top \mid H \in \Delta\}$ is $\mathcal{M}_2(\Delta)$, then an MGAS of $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$ is $\mathcal{M}_2(\emptyset)$. Then the literals in $\mathfrak{L}_{\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle} \cap \mathcal{A}^*$ are never positive and Π' is an ordinary extended logic program that coincides with Π_1. Therefore, $\Pi_1 \oslash \Pi_2 \equiv \Pi_1 \cup \Pi_2$.

⊘-SP-4, Idempotence: $\Pi \oslash \Pi \equiv \Pi$.

Suppose Π has answer sets. Then, $\Pi \cup \Pi$ does too (namely the same). By Strong Consistency, $\Pi \oslash \Pi \equiv \Pi \cup \Pi \equiv \Pi$. Suppose Π does not have answer sets. Then $\Pi \oslash \Pi$ has the update program $\langle\, \Pi' \cup \Pi, \mathcal{A}^* \,\rangle$ that does not have generalised answer sets either. Thus, $\Pi \equiv \Pi \oslash \Pi$.

⊘-SP-9, Weak Irrelevance of Syntax: Let Π, Π_1, and Π_2 be logic programs under the same language. If $T_{\mathcal{N}_2}(\Pi_1) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_2)$ then $\Pi \oslash \Pi_1 \equiv \Pi \oslash \Pi_2$.

Suppose $T_{\mathcal{N}_2}(\Pi_1) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_2)$, and each $\Pi \cup \Pi_1$ and $\Pi \cup \Pi_2$ have at least an answer set. By Strong Consistency, $\Pi \oslash \Pi_1 \equiv_{\mathsf{ASP}} \Pi \cup \Pi_1$ and $\Pi \oslash \Pi_2 \equiv_{\mathsf{ASP}} \Pi \cup \Pi_2$. Thus, $T_{\mathcal{N}_2}(\Pi \cup \Pi_1) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi \cup \Pi_2)$. Therefore, if $T_{\mathcal{N}_2}(\Pi_1) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_2)$, then $\Pi \oslash \Pi_1 \equiv_{\mathsf{ASP}} \Pi \oslash \Pi_2$.

⊘-SP-7, Augmented Update: If $\Pi_1 \subseteq \Pi_2$ then $\Pi_1 \oslash \Pi_2 \equiv \Pi_2$.

Suppose $\Pi_1 \subseteq \Pi_2$ and that both Π_1 and Π_2 have answer sets. This means that $\Pi_1 \cup \Pi_2 = \Pi_2$. By Strong Consistency, $\Pi_1 \oslash \Pi_2 = \Pi_2$. Suppose $\Pi_1 \subseteq \Pi_2$ and that at least one of them has no answer sets. By $\Pi_1 \cup \Pi_2 = \Pi_2$, Π_2 never has answer sets. Thus, the update program $\langle\, \Pi_1' \cup \Pi_2, \mathcal{A}^* \,\rangle$ never has generalised answer sets. In each case, $\Pi_2 \equiv \Pi_1 \oslash \Pi_2$.

⊘-SP-6, Non-interference: If Π_1 and Π_2 are programs defined over *disjoint alphabets*, and either both of them have answer sets or do not, then $\Pi_1 \oslash \Pi_2 \equiv \Pi_2 \oslash \Pi_1$.

Assume that Π_1 and Π_2 are defined over disjoint alphabets and that both Π_1 and Π_2 have at least an answer set. Then, $\Pi_1 \cup \Pi_2$ has at least an answer set too. Thus, by Strong Consistency, $\Pi_1 \oslash \Pi_2 \equiv \Pi_1 \cup \Pi_2 \equiv \Pi_2 \cup \Pi_1 \equiv \Pi_2 \oslash \Pi_1$. Now suppose that both Π_1 and Π_2 have no answer sets. Then, the update program $\langle\, \Pi_1' \cup \Pi_2, \mathcal{A}^* \,\rangle$ never has generalised answer sets. Thus, $\Pi_1 \oslash \Pi_2 \equiv \Pi_2 \oslash \Pi_1$ in either case.

□

These are the properties considered as fundamental for a proper semantics for updates. Further properties are left to a more general semantics. Moreover, Appendix A summarises semantics versus properties, seen along this thesis.

Following the reader can find a general formal description of an implementation of this operator, as well as the implementation itself at `http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html`.

6.3 Computing Updates with ODLP

As an important element of Logic Programming that distinguishes it over other theoretical approaches, this section comprises both foundation and software tools of the implementation of this semantics, coming from *Ordered-disjunctive Logic Programming*, introduced in Section 3.5. [1]

Ordered-disjunctive Logic Programming, ODLP by [Brewka, 2002; Brewka et al., 2004], is an extension to ELP's and may be defined in broad way as follows: a simple *ordered-disjunction program* is a set of rules of form

$$C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_k$$

where C_i, A_j and B_l are all ground literals. $C_1, \ldots, C_n$ are usually named the *choices of a rule* and their intuitive reading is as follows: The ordered disjunction is used only in rule heads to select some of the answer sets of a program as the preferred ones. If C_1 is possible, then C_1; if C_1 is not possible, then try C_2; ...; if neither $C_i, \ldots, C_{n-1}$ is possible then try C_n. Moreover, one may identify some special cases such as: if $n = 0$ the rule is a *constraint*; and finally, facts are those rules where $m = k = 0$. In the particular case of this dissertation, the required codification of ordered disjunctive programs shall be just $n = 2, m = k = 0$, as in Section 6.3.5.

6.3.1 ODLP-reduct

In particular, there is a reduct of ODLP-rules with respect to a set of literals, as well as a reduct of ODLP-programs:

Definition 6.3 ($\times$-reducts, [Brewka et al., 2004])**.** *Let*

$$\rho = C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_k$$

be a rule and $\mathcal{M}$ a set of literals. Then, the $\times$-reduct $\rho_\times^\mathcal{M}$ of ρ is defined as

$$\rho_\times^\mathcal{M} = \{C_i \leftarrow A_1, \ldots, A_m \mid C_i \in \mathcal{M} \text{ and } \mathcal{M} \cap \{C_1, \ldots, C_{i-1}, B_1, \ldots, B_k\} = \emptyset\}.$$

[1] For complete formal definitions on ODLP please refer to Section 3.5.

Let Π *be an* ODLP. *The* $\times$*-reduct* $\Pi_{\times}^{\mathcal{M}}$ *of* Π *is defined as:*

$$\Pi_{\times}^{\mathcal{M}} = \bigcup_{\rho \in \Pi} \rho_{\times}^{\mathcal{M}}.$$

One may compute the *answer sets of a* $\times$*-reduct,* $\Pi_{\times}^{\mathcal{M}}$ as possible interpretations of an ODLP, by means of the following proposition.

Proposition 6.1 ([Brewka et al., 2004])**.** *Let* Π *be an* ODLP *and* $\mathcal{M}$ *a set of literals. Then,* $\mathcal{M}$ *is an answer set of* Π *if and only if the following three conditions hold:*

1. $\mathcal{M} = Cn(\Pi_{\times}^{\mathcal{M}})$,
2. $\mathcal{M}$ *is consistent, and*
3. $\mathcal{M}$ *satisfies every rule* $\rho \in \Pi$.

Finally, the models of an ODLP are defined in terms of *preferred answer sets* in at least three categories with respect to its *satisfaction degree*. Namely, *cardinality-preference*, *inclusion-preference* and *paretto-preference*. In this dissertation, the focus is on the first two of them, as later explained in Section 6.3.5 and Section 8.4.1.

6.3.2 ODLP-semantics

Before introducing semantics to characterise ODLP's, it is necessary to define what a *satisfaction degree* is.

Definition 6.4 (Degree of satisfaction, $\mathsf{d}_{\mathcal{M}}(\rho)$ [Brewka et al., 2004])**.** *Let* $\mathcal{M}$ *be an answer set of an* ODLP. *The satisfaction degree* $\mathsf{d}_{\mathcal{M}}(\rho)$ *by* $\mathcal{M}$ *of a rule* ρ *of form*

$$C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_k$$

- *is* 1 *if* $A_j \notin \mathcal{M}$, *for some* j, *or* $B_i \in \mathcal{M}$ *for some* i;
- *is* j *with* $1 \leq j \leq n$, *if all* $A_j \in \mathcal{M}$, *no* $B_i \in \mathcal{M}$, *and*

$$j = \min\{r \mid C_r \in \mathcal{M}\}$$

With the degree of satisfaction, one can define many *preference relations*, as the ones suggested by Brewka et al.. For that purpose, they also define the set of rules with a degree of satisfaction as follows: Given a set of literals $\mathcal{M}$, let $\mathcal{M}^i(\Pi) = \{\rho \in \Pi \mid \mathsf{d}_{\mathcal{M}}(\rho) = i\}$.

Definition 6.5 (Cardinality Preference [Brewka et al., 2004]). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be answer sets of an ODLP, Π. Then, $\mathcal{M}_1$ is cardinality-preferred to $\mathcal{M}_2$ (denoted as $\mathcal{M}_1 >_c \mathcal{M}_2$) if and only if there is an i such that $|\mathcal{M}^i(\Pi)_1| >_c |\mathcal{M}^i(\Pi)_2|$; and for all $j < i$ and $|\mathcal{M}^j(\Pi)_1| = |\mathcal{M}^j(\Pi)_2|$.*

In other words, model $\mathcal{M}_1$ is *cardinality-preferred* to $\mathcal{M}_2$ from program Π if and only if there is a *satisfaction degree* for which a set of rules satisfied by $\mathcal{M}_1$ is bigger than the set of rules satisfied by $\mathcal{M}_2$. The inclusion preference has a similar intuition:

Definition 6.6 (Inclusion Preference [Brewka et al., 2004]). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be answer sets of an ODLP, Π. Then, $\mathcal{M}_1$ is inclusion-preferred to $\mathcal{M}_2$ ($\mathcal{M}_1 >_i \mathcal{M}_2$) if and only if there is an i such that $\mathcal{M}^i(\Pi)_2 \subset \mathcal{M}^i(\Pi)_1$; and for all $j < i$ and $\mathcal{M}^j(\Pi)_1 = \mathcal{M}^j(\Pi)_2$.*

Brewka et al. also state the following important relation.

Proposition 6.2 (Brewka et al.). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be answer sets of an ODLP, Π. Then $\mathcal{M}_1 >_i \mathcal{M}_2$ implies that $\mathcal{M}_1 >_c \mathcal{M}_2$.*

Finally, the indented models of an ODLP are called *k-preferred answer sets* and are based on the satisfaction degree and the preference criterion. Formally,

Definition 6.7 (Preferred Answer Sets, Brewka et al.). *A set of literals $\mathcal{M}$ is a k-*preferred answer set *(where $k \in \{c, i\}$) of an ODLP, Π, if and only if $\mathcal{M}$ is an answer set of Π and there is no answer set $\mathcal{M}'$ of Π such that $\mathcal{M}' >_k \mathcal{M}$.*

There are more preference criterions, but the two definitions just introduced are the most relevant to this dissertation.

6.3.3 ODLP and Weak Constraints

There is a very-interesting alternative to preferred answer sets by means of weak constraints. For this end, Brewka et al. have found a relation between them. They propose a translation of a normal logic program Π with weak constraints into an ODLP, Π', by replacing every weak constraint ω of form (3.6) —with $w = p = 1$— into the following pair of rules:

$$\alpha_\omega \times \sim\alpha_\omega \leftarrow \top \tag{6.2}$$

$$\bot \leftarrow \alpha_\omega, b_1, \ldots, b_k, \neg b_{k+1}, \ldots \neg b_m \tag{6.3}$$

where α_ω is a new atom that denotes the constraint ω is not violated.

Accordingly, Brewka et al. state that *ω-preference* corresponds to *c-preference* because the amount of violated weak constraints is exactly the same amount of rules satisfied to the second degree [Brewka et al., 2004]. Such correspondence may be useful for the implementations of both Section 6.3.5 and Section 8.4.2, in upcoming chapters.

6.3.4 ODLP-solver

Finally, as one of the distinguishing features of logic programming, it is very important to notice that `PSmodels`[1] is an *ODLP implemented prototype* [Brewka et al., 2002], which consists of an extension to `SMODELS`[2] [Niemela and Simons, 1997] to compute *preferred stable models* of normal logic programs. However, it is also important to point out that the sources themselves need some *debugging*[3].

PSmodels, in a bug-free implementation however, should compute *preferred stable models* (also known as *k-preferred answer sets*) of normal logic programs under ODLP and should be considered when thinking of implementing the update semantics proposed in this dissertation.

One of the features of `PSmodels` is to tell how many times the test program has been invoked to check whether a stable model of a given ODLP is a preferred one. Besides the original implementation sources available at the author's sites, there is a compiled *online front-end* running at `http://www.in.tu-clausthal.de/~guadarrama/updates/psmodels.html` that provides the original compiled `PSmodels` to run as an *Internet service* in a webpage, rather than the classical online run. This overcomes the need to download it, compile it and run it locally.

In order to compute the MGAS's of an abductive logic program, there is a translation from [Osorio et al., 2004] to realise it in ODLP and the translation goes as follows.

6.3.5 Translating into ODLP

The following function is a version of the one by Osorio et al. in my own notation, that takes an abductive logic program and translates it into an ordered-disjunctive one.

Definition 6.8 (Ordered Translation, $\mathcal{O}$ [Osorio et al., 2004])**.** *Let* $\langle \Pi, \mathcal{A}^* \rangle$ *be an abductive logic program. A* translation into an ordered program*, denoted as* $\mathcal{O}(\Pi, \mathcal{A}^*)$*, consists of the following. For any literal* $\ell \in \mathcal{A}^*$*, the clause* ρ_α *is a rule of the form* $\alpha' \times \alpha \leftarrow \top$*, where* α' *is a literal that does not occur in the original abductive program. Then,* $\mathcal{O}(\Pi, \mathcal{A}^*) = \Pi \cup \{\rho_\alpha \mid \alpha \in \mathcal{A}^*\}$.

The intuition behind this is to take an abductive program and to append the abductive atoms (in a form of ODLP rules) to its ELP program. Then, the resulting union is a regular ODLP program. The following example, borrowed from [Osorio et al., 2004] who in turn was inspired by Balduccini and Gelfond's, illustrates the just defined translation.

[1] The sources may be downloaded from `http://www.tcs.hut.fi/Software/smodels/priority` and there is a graphical user front end at `http://www.in.tu-clausthal.de/~guadarrama/updates/psmodels.html` that allows to execute preferred logic programs online.

[2] This solver may be downloaded from `http://www.tcs.hut.fi/Software/smodels/` and run via online with a graphical user front end at `http://www.in.tu-clausthal.de/~guadarrama/updates/smodels.html`.

[3] For instance, Version 2.26a will crash with a simple program like $\{a.\}$.

Example 6.3 ([Osorio et al., 2004]). *Suppose the abductive program* $\langle\, \Pi, \{q, s, t\}\,\rangle$ *where*

$$\begin{aligned}\Pi = \{p &\leftarrow \neg q\\ r &\leftarrow \neg s\\ q &\leftarrow t\\ s &\leftarrow t\\ \bot &\leftarrow p, r\}\end{aligned}$$

The corresponding ordered program $\mathcal{O}(\Pi, \{q, s, t\})$ *is then*

$$\begin{aligned}\{q' \times q &\leftarrow \top\\ s' \times s &\leftarrow \top\\ t' \times t &\leftarrow \top\\ p &\leftarrow \neg q\\ r &\leftarrow \neg s\\ q &\leftarrow t\\ s &\leftarrow t\\ \bot &\leftarrow p, r\}\end{aligned}$$

with three ***preferred answer sets****:* $\{s, t, q, q', s'\}; \{r, q, s', t'\}; \{p, s, q', t'\}$. *Note that* Π *is inconsistent. However, the new ordered program is now consistent.*

Last, the generalisation of this translation proves to be correct, with a slight correction of a typo from the original lemma:

Lemma 6.1 ([Osorio et al., 2004]). $\mathcal{M} \cap \mathfrak{L}_\Pi$ *is a* generalised answer set *of an abductive program* $\langle\, \Pi, \mathcal{A}^*\,\rangle$ *if and only if* $\mathcal{M}$ *is a preferred answer set of* $\mathcal{O}(\Pi, \mathcal{A}^*)$

and the validity of set inclusion in ODLP:

Theorem 6.2 ([Osorio et al., 2004]). *Let* $\langle\, \Pi, \mathcal{A}^*\,\rangle$ *be an abductive program and* $\mathcal{M}$ *a set of atoms.* $\mathcal{M} \cap \mathfrak{L}_\Pi$ *is a* minimal generalised answer set *of the abductive program if and only if* $\mathcal{M}$ *is an* i-preferred answer set *of* $\mathcal{O}(\Pi, \mathcal{A}^*)$.

Now it is easy to see how to use ODLP to update an extended logic program with $\oslash$ operation.

6.3.6 Updating with ODLP

Finally, this translation proves to be useful for the context of updates in MGAS with the following formalisation.

Definition 6.9 (Ordered Disjunctive Update Program)**.** *Given an update program* $\mathbf{\Pi}_{\oslash} = \Pi_1 \oslash \Pi_2$ *over a set of atoms* $\mathcal{A}$*, and its corresponding abductive program* $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$*, its* Ordered-disjunctive Update Program *corresponds to*

$$\mathcal{O}(\Pi' \cup \Pi_2, \mathcal{A}^*)$$

The following examples illustrate how this translation works and how it can be computed with ODLP.

Example 6.4 (continued)**.** *Consider Example 6.1 again, with its corresponding abductive program* $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$*, whose ODLP transformation, by Definition 6.8, consists of*

$$\begin{aligned}\mathcal{O}(\Pi' \cup \Pi_2, \mathcal{A}^*) = \{&\alpha'_1 \times \alpha_1 \leftarrow \top \qquad \alpha'_2 \times \alpha_2 \leftarrow \top \\ &\alpha'_3 \times \alpha_3 \leftarrow \top \qquad \alpha'_4 \times \alpha_4 \leftarrow \top\end{aligned}$$

$$\begin{aligned} \mathit{sleep} &\leftarrow \neg \mathit{tv}(\mathit{on}), \neg \alpha_1 \\ \mathit{night} &\leftarrow \neg \alpha_2 \\ \mathit{watch}(\mathit{tv}) &\leftarrow \mathit{tv}(\mathit{on}), \neg \alpha_3 \\ \mathit{tv}(\mathit{on}) &\leftarrow \neg \alpha_4 \\ {\sim}\mathit{tv}(\mathit{on}) &\leftarrow \mathit{power}(\mathit{failure}) \\ \mathit{power}(\mathit{failure}) &\leftarrow \top\} \end{aligned}$$

By Theorem 6.2, the minimal generalised answer sets of every abductive program $\langle\, \Pi, \mathcal{A}^* \,\rangle$ correspond to the intended models of some ordered disjunctive program Π' that can be easily run on a computer. As a result, the unique **preferred answer set** of such an ODLP program,

$$\{\mathit{sleep}, \alpha'_1, \mathit{night}, \alpha'_2, \alpha'_3, \alpha_4, {\sim}\mathit{tvon}, \mathit{pfailure}\}$$

coincides with our intuition.

Another interesting experiment is one that has to do with an update with *vacuous information.*

Example 6.5 (continued)**.** *Consider Example 1.1 and Example 6.2 again. The trans-*

lation in this particular case corresponds to the following ordered disjunctive program.

$$\mathcal{O}(\Pi' \cup \Pi_2, \mathcal{A}^*) = \{\alpha'_1 \times \alpha_1 \leftarrow \top \qquad \alpha'_2 \times \alpha_2 \leftarrow \top$$
$$\alpha'_3 \times \alpha_3 \leftarrow \top \qquad \alpha'_4 \times \alpha_4 \leftarrow \top$$

$$\begin{aligned} day &\leftarrow \neg night, \neg\alpha_1 \\ night &\leftarrow \neg day, \neg\alpha_2 \\ see(stars) &\leftarrow night, \neg cloudy, \neg\alpha_3 \\ \sim see(stars) &\leftarrow \neg\alpha_4 \\ stars &\leftarrow constellations \\ constellations &\leftarrow stars\} \end{aligned}$$

By executing this program in `PSmodels` on inclusion-based criterion, its sole **preferred answer set** is $\{\alpha'_1, \alpha'_2, day, \alpha'_3, \alpha'_4, \sim see(stars)\}$, which coincides with our intuition.

As an expected result, this kind of translation led to an implemented system[1] at `http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html`. See also Section B.1 for further alternatives.

6.4 Conclusions for Chapter 6

This chapter is a preliminary study of one of the major drawbacks in existing update semantics of ASP programs and comprises a *relaxation method* to overcome them. Such a drawback is mainly due to the strong *syntax-oriented techniques* employed in other approaches. As a result, this chapter has introduced a basic set of relevant properties to be met, in particular *Weak Irrelevance of Syntax* and *Strong Consistency*. In addition to that, the chapter includes a method for *relaxing an original knowledge base* so that it is flexible-ehough to cope with new possibly-conflicting information upcoming from a changing environment.

In particular, the proposed method produces a new *resulting knowledge base* of merging the relaxed one with the update, which is an *abductive logic program*. Out of the new abductive knowledge-base, one can draw a minimal set of conclusions, according to a *minimal-change principle* and to a *preference relation* between the relaxed knowledge-base and the update.

One of the existing mechanisms to interpret the resulting knowledge base is to compute the MGAS's of the merged base, which can also be translated into an ODLP preference program, here introduced. In consequence, the use of ODLP is three-fold: it

[1] This implementation is due to the student B. Fuhrmann.

can be used to compute MGAS's, abductive programs, and updates of logic programs, which is of great interest for its proposed solver.

Finally, this method proves to overcome problems of *syntax-oriented approaches* for its *model-orientation*, satisfying key properties that preserve the *logical contents* of knowledge bases. Such properties and operational semantics make the framework a strong foundation for the rest of the thesis.

Chapter 7

Update Sequences

7.1 Introduction

A traditional and general goal of belief updates[1] is dealing with contradictory information or with new data from a changing environment. As a result, several proposals (refer to Chapter 4) have been formulated to update sequences of logic programs. According to these semantics, knowledge is given by a sequence of logic programs where each program is considered an update of the previous one.

All of these semantics are based on the notion of *causal rejection of rules* —Section 4.2— , which enforces that, in case of conflicts between rules, *more recent rules override older ones* [Alferes et al., 2005, 1999; Eiter et al., 2000a]. However, there are particular rare challenging (even so possible) situations that might lead to *counterintuitive models* of knowledge that have been subject of recent research and matter of formulation of new principles. As a preliminary result to solve these particular situations, Guadarrama; Guadarrama et al.; Osorio and Zacarías have proposed approaches based upon the *logical contents* of programs rather than the *syntactical causal rejection of rules*, and that approach is the main core of Chapter 6. Nevertheless, despite the capability to overcome counterintuitive results of *unforeseen situations*, the main drawback of the semantics pointed out in that chapter is the *limitation to only one update*.

In need of a correct way to represent *dynamic knowledge*, some authors like Eiter et al.; Sakama and Inoue; Zhang and Foo made their foundations on *Answer Set Programming* (See Section 3.1) —or simply ASP— for being one of the most solid and studied semantics for logic programs up to the last decades, as introduced in Chapter 4. In addition and as an alternative, Alferes et al. proposed an approach of DyLP on *Well-founded Semantics* (WFS) [Alferes et al., 2005, 1999], which may be less complex than ASP (see [Dix, 1995b] for a deeper comparison), but also less *representative*.

The original problem discussed earlier in Chapter 6 is that most of the mentioned approaches are founded on their *causal rejection principle* that leads to *counterintuitive*

[1] Notice that in practice, computing researchers call it *updates* although the difference with *belief revision* is both technical and philosophical! See Section 2.4 and Section 7.3 for further details.

behaviour under certain circumstances, like *redundancy* or *tautological updates*. In order to illustrate this claim, consider the following theory, proposed to solve a very similar problem in [Alferes et al., 2005], describing a particular situation of the sky.

Example 7.1. *[From Example 1.1]*

Suppose an agent who believes that when *it is* day *it is not* night *and vice versa, and that there are* stars *when it is* night *and when there are no* clouds. *Finally, that at the current moment it is a fact that there are no* stars. *This simple story may be coded*[1] *into* Π_1 *as follows:*

$$\begin{aligned} \Pi_1 = \{ day &\leftarrow \neg night \\ night &\leftarrow \neg day \\ stars &\leftarrow night, \neg cloudy \\ \sim stars &\leftarrow \top \} \end{aligned}$$

whose unique answer set is $\{day, \sim stars\}$. *Later, the agent acquires new information stating that* stars *and* constls *(*constellations*) are the same thing, as coded in* Π_2. *As soon as the agent updates* Π_1 *with program*

$$\begin{aligned} \Pi_2 = \{ stars &\leftarrow constls \\ constls &\leftarrow stars \} \end{aligned}$$

the expanded alphabet *of the two programs contains only one new* extra atom *with respect to* Π_1*: constls. As the model of* Π_2 *is obviously the empty answer set, constls is considered* synonym *of stars by means of* Π_2*, and thus the update should not change the* original beliefs. *However, the update yields an* extra answer set *in some of the existing update semantics based on the* causal rejection principle *—Section 4.2:*

$$\{stars, constls, night\}$$

which does not coincide with common intuition.

The reason is that, although stars can not be true, introducing constls gives another possibility for stars to be true. Thus, the additional answer set is implied.

In general, these supplementary rules in the update are a conservative extension *[Osorio et al., 2001] to* Π_1*: the original language is extended and all answer sets ought to be extensions of the old answer sets. In this specific situation, constls should be true*

[1] Notice that there are other ways to represent the story. The problem is, however, what to do in this particular situation, when the agent runs across this piece of information.

if and only if stars is true.

A solution to the problem is a semantics based upon *Generalised Answer Sets* — introduced in Section 3.6— that satisfies several structural properties, overcoming the above problems from the proposals analysed in Chapter 4.

In this chapter, the reader can find a particular case of the framework introduced in Chapter 6 to perform sequences of updates, that overcomes the referred disadvantages of the causal-rejection principle. In addition, this chapter includes a general description to implement update sequences in Section 7.6 that leads to a running prototype, as well as tools, methods, and directions to get the running solver itself, preceded by the definition of the semantics in Section 7.2 and properties in Section 7.3. Finally, the chapter ends with concluding remarks in Section 7.7.

7.2 ⊗-Operation

Intuitively, this approach consists in relaxing all rules in previous programs to the latest update, with a unique abducible. As a result, a transformed *relaxed program* is part of an abductive program, and the other part is the set of abducibles. Out of the corresponding GAS's of the abductive program, one may get the minimal with respect to its *sequence order* —MSGAS. Finally, the expected model should expressed in its original alphabet out of the MSGAS's.

An *α-relaxed rule* is a rule ρ that is *weakened* by a default-negated atom α in its body: $\mathsf{Head}(\rho) \leftarrow \mathsf{Body}(\rho) \cup \{\neg\alpha\}$. In addition, an *$\alpha$-relaxed program* is a set of α-relaxed rules.

In the particular case of sequences, there are both relaxed sequences and a relaxed program, as formally expressed below.

The *α-relaxed program* of a sequence of ELP's, $(\Pi_1, \Pi_2, \ldots, \Pi_n)$, over a set of atoms $\mathcal{A}$, is the set $\Pi' = \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_{n-1}$, where Π'_i is the *α-relaxed program* of Π_i by a new unique abducible $\alpha \notin \mathcal{A}$ for each rule $\rho \in \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_{n-1}$ with $1 \leq i \leq n-1$, and $(\Pi'_1, \Pi'_2, \ldots, \Pi'_{n-1})$ is the corresponding *α-relaxed sequence*.

Moreover, it is necessary to set up an abductive program from the α-relaxed program of an update sequence and establish an order to get the desired properties. The problem consists in favouring those models that have the least number of abducibles at the highest level of the sequence.

The functions introduced in Section 3.4 from DLV's *weak constraints* have more general characteristics and are a good candidate to figure out the problem. Its general intuition consists in extending ASP to include constraints that may be violated. Accordingly, the goal of such an extended program is the optimal model(s) that violates the minimal number of weak constraints at a certain *priority level*.

The goal of the main *objective function*, as called by Leone et al., is to simplify the combination of weights in priority levels.

This function has a strong relation to GAS, as shown in Section 8.4.2. As a result, it can be a basis to bring about a *cardinality order*, and simplified to the needs of update

sequences as follows.

Definition 7.1 (Abductive Sequence Order; MSGAS)**.** *Given an update sequence of ELP's,* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$*, over a set of atoms* $\mathcal{A}$ *with* n *as an integer; with its corresponding* α*-relaxed sequence* $(\Pi'_1, \Pi'_2, \ldots, \Pi'_{n-1})$*, where* $\alpha \in \mathcal{A}^*$ *and the* α*-relaxed program* Π'*; and the abductive program* $\Pi_{\mathcal{A}^*} = \langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle$*:*

- $w(l) = \begin{cases} 1, & l = 1 \\ w(l-1) \cdot |\mathcal{A}^*| + 1 & l > 1 \end{cases}$ *where* l *is a positive integer.*
- $s(\Delta) = \sum_{l=1}^{n-1}(w(l) \cdot |\{\alpha \mid \alpha \in \mathfrak{L}_{\Pi'_l}\}|)$*, where* $\mathcal{M}(\Delta)$ *is a GAS of* $\Pi_{\mathcal{A}^*}$ *and* $\alpha \in \Delta$*.*
- $\mathcal{M}(\Delta_1) \leq_{\mathcal{S}} \mathcal{M}(\Delta_2)$ *if and only if* $s(\Delta_1) \leq s(\Delta_2)$*.*
- $\mathcal{M}(\Delta_1) \equiv_{\mathcal{S}} \mathcal{M}(\Delta_2)$ *if and only if* $s(\Delta_1) = s(\Delta_2)$*.*
- $\mathcal{M}(\Delta)$ *is a* Minimal Sequenced Generalised Answer Set*, MSGAS, of* $\Pi_{\mathcal{A}^*}$ *if and only if* $\mathcal{M}(\Delta)$ *is minimal with respect to* $\leq_{\mathcal{S}}$*.*

Let us illustrate this definition with the following simple sequence of updates:

Example 7.2. *Suppose the sequence* $\Pi_1 \otimes \Pi_2 \otimes \Pi_3$ *where*

$$\begin{aligned} \Pi_1 = &\quad \{(a \leftarrow \top), (b \leftarrow \top)\} \\ \Pi_2 = &\quad \{\sim b \leftarrow \top\} \\ \Pi_3 = &\quad \{b \leftarrow b, \neg c\} \end{aligned}$$

Its α*-relaxed sequence*

$$\begin{aligned} \Pi'_1 = &\quad \{(a \leftarrow \neg\alpha_1), (b \leftarrow \neg\alpha_2)\} \\ \Pi'_2 = &\quad \{\sim b \leftarrow \neg\alpha_3\} \end{aligned}$$

and its α*-relaxed program*

$$\begin{aligned} \Pi' = &\quad \{(a \leftarrow \neg\alpha_1), (b \leftarrow \neg\alpha_2), \\ &\qquad \sim b \leftarrow \neg\alpha_3\} \end{aligned}$$

The abductive program $\langle \Pi' \cup \Pi_3, \mathcal{A}^* \rangle$ *has the following* GAS*'s:*

$$\begin{aligned} \mathcal{M}(\Delta_1) &= \{a, \sim b\}_{\{\alpha_2\}}; & \mathcal{M}(\Delta_2) &= \{\sim b\}_{\{\alpha_1,\alpha_2\}}; \\ \mathcal{M}(\Delta_3) &= \{a, b\}_{\{\alpha_3\}}; & \mathcal{M}(\Delta_4) &= \{b\}_{\{\alpha_1,\alpha_3\}}; \\ \mathcal{M}(\Delta_5) &= \{a\}_{\{\alpha_2,\alpha_3\}}; & \mathcal{M}(\Delta_6) &= \{\}_{\{\alpha_1,\alpha_2,\alpha_3\}} \end{aligned}$$

where

$$\begin{aligned} \Delta_1 &= \{\alpha_2\}; & \Delta_2 &= \{\alpha_1, \alpha_2\}; \\ \Delta_3 &= \{\alpha_3\}; & \Delta_4 &= \{\alpha_1, \alpha_3\}; \\ \Delta_5 &= \{\alpha_2, \alpha_3\}; & \Delta_6 &= \{\alpha_1, \alpha_2, \alpha_3\} \end{aligned}$$

and each w *has the following weights:*

$$w(1) = 1; w(2) = 4$$

and the corresponding weights for each Δ *are*

$$\begin{aligned} s(\Delta_1) &= 1; & s(\Delta_2) &= 2; \\ s(\Delta_3) &= 4; & s(\Delta_4) &= 5; \\ s(\Delta_5) &= 5; & s(\Delta_6) &= 6 \end{aligned}$$

According to the sequence, the order of these GAS*'s is*

$$\mathcal{M}(\Delta_1) \leq_{\mathcal{S}} \mathcal{M}(\Delta_2) \leq_{\mathcal{S}} \mathcal{M}(\Delta_3) \leq_{\mathcal{S}} \mathcal{M}(\Delta_4) \leq_{\mathcal{S}} \mathcal{M}(\Delta_5) \leq_{\mathcal{S}} \mathcal{M}(\Delta_6)$$

Note that $\mathcal{M}(\Delta_5) \leq_{\mathcal{S}} \mathcal{M}(\Delta_4)$ *holds as well! Then,* $\mathcal{M}(\Delta_4) \equiv_{\mathcal{S}} \mathcal{M}(\Delta_5)$.

Finally, $\{a, \sim b\}_{\{\alpha_2\}}$ *is its unique* MSGAS.

Strictly speaking, $\leq_{\mathcal{S}}$ is a *pre-order relation* (also known as *preorder* and *quasiorder*) because it is reflexive and transitive. However, $\leq_{\mathcal{S}}$ is mapped into a total-order relation of natural numbers that represent the weight of a preferred model —$s(\Delta)$— as shown in Definition 8.6.

Intuitively, these orders are with respect to the latest update, postulates $(\mathsf{R} * 1)$ and $(\mathsf{U} \circledast 1)$ in Tables 2.5 and 2.6 and to a *minimal change with respect to cardinality*: MSGAS. Such postulates are to be recapitulated later in depth, in Section 8.3.4.

Proposition 7.1. *Abductive Sequence Order* ($\leq_{\mathcal{S}}$) *is not an antisymmetric relation.*

Proof. Consider the update sequence $\Pi_1 \otimes \Pi_2 \otimes \Pi_3$ from Example 7.2, where two of its

GAS's are $\mathcal{M}(\Delta_4) = \{b\}_{\{\alpha_1,\alpha_3\}}$ and $\mathcal{M}(\Delta_5) = \{a\}_{\{\alpha_2,\alpha_3\}}$. Clearly, $\mathcal{M}(\Delta_4) \leq_{\mathcal{S}} \mathcal{M}(\Delta_5)$ and $\mathcal{M}(\Delta_5) \leq_{\mathcal{S}} \mathcal{M}(\Delta_4)$. However, $\mathcal{M}(\Delta_4) \neq \mathcal{M}(\Delta_5)$.

□

From Proposition 7.1 one can easily verify the following result.

Corollary 7.1. *Abductive Sequence Order ($\leq_{\mathcal{S}}$) is a total pre-order relation.*

These are the necessary definitions to formulate an updating sequence of logic programs, proposed in this chapter. In short, the formulation consists in the transformation of the update sequence into a single abductive program for which there is a set of *preferred abducibles* according to its order in the relaxed sequence. Finally, *Update Answer Sets* are the models one expects from an updating sequence:

Definition 7.2 ($\otimes$-update Answer Set)**.** *Given a $\otimes$-sequence of updating ELP's $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$, with $n \geq 2$, over a set of atoms $\mathcal{A}$, the set $\mathcal{S} \subseteq \mathcal{A}$ is its* $\otimes$-update answer set *if and only if $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ for some* minimal sequenced generalised answer set *$\mathcal{S}'$ of the sequence, and $\otimes$ is the corresponding* update operator.

Let us illustrate the entire framework with a couple of more examples.

Example 7.3. *Suppose an update to Π_1 with Π_2, with Π_3 with Π_4 and with Π_5, where $\Pi_1 = \{b \leftarrow \top\}; \Pi_2 = \{\sim b \leftarrow \top\}; \Pi_3 = \{b \leftarrow b, \neg c\}; \Pi_4 = \{a \leftarrow \top\}; \Pi_5 = \{\sim a \leftarrow \top\}$, from which one would expect $\{\sim a, \sim b\}$ as its* ***unique update answer set****. The abductive program of the sequence corresponds to $\langle\, \Pi' \cup \Pi_5, \mathcal{A}^* \,\rangle$, where $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ and*

$$\begin{aligned} \Pi' = \{b &\leftarrow \neg\alpha_1 && (7.1)\\ \sim b &\leftarrow \neg\alpha_2 && (7.2)\\ b &\leftarrow b, \neg c, \neg\alpha_3 && (7.3)\\ a &\leftarrow \neg\alpha_4\} && (7.4) \end{aligned}$$

whose GAS's are

$$\begin{aligned} \mathcal{M}(\Delta_1) &= \{\sim a, \sim b\}_{\{\alpha_1,\alpha_4\}}; & \mathcal{M}(\Delta_2) &= \{\sim a, \sim b\}_{\{\alpha_1,\alpha_3,\alpha_4\}};\\ \mathcal{M}(\Delta_3) &= \{\sim a, b\}_{\{\alpha_2,\alpha_4\}}; & \mathcal{M}(\Delta_4) &= \{\sim a, b\}_{\{\alpha_2,\alpha_3,\alpha_4\}};\\ \mathcal{M}(\Delta_5) &= \{\sim a\}_{\{\alpha_1,\alpha_2,\alpha_4\}}; & \mathcal{M}(\Delta_6) &= \{\sim a\}_{\{\alpha_1,\alpha_2,\alpha_3,\alpha_4\}} \end{aligned}$$

and

$$\begin{aligned} \Delta_1 &= \{\alpha_1, \alpha_4\}; & \Delta_2 &= \{\alpha_1, \alpha_3, \alpha_4\}; \\ \Delta_3 &= \{\alpha_2, \alpha_4\}; & \Delta_4 &= \{\alpha_2, \alpha_3, \alpha_4\}; \\ \Delta_5 &= \{\alpha_1, \alpha_2, \alpha_4\}; & \Delta_6 &= \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\} \end{aligned}$$

where each w has the following weights:

$$w(1) = 1; w(2) = 5; w(3) = 21; w(4) = 85$$

and the corresponding weights for each Δ are

$$\begin{aligned} s(\Delta_1) &= 86; & s(\Delta_2) &= 107; \\ s(\Delta_3) &= 90; & s(\Delta_4) &= 111; \\ s(\Delta_5) &= 91; & s(\Delta_6) &= 112 \end{aligned}$$

*Next, its unique MSGAS clearly is $\mathcal{M}(\Delta_1)$, and last, its **update answer set** is just $\{\sim a, \sim b\}$, as one would expect.*

Finally, the problem introduced in Example 1.1recapitulated in Example 7.1 and solved in Example 6.2, can easily be modelled with this current sequencial approach:

Example 7.4. *Consider the update sequence $\Pi_1 \otimes \Pi_2$ from Example 7.1, whose abductive program may be coded into $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$.*

$$\begin{aligned} \Pi' = \{\, \mathit{day} &\leftarrow \neg \mathit{night}, \neg\alpha_1 \\ \mathit{night} &\leftarrow \neg \mathit{day}, \neg\alpha_2 \\ \mathit{stars} &\leftarrow \mathit{night},\ \neg \mathit{cloudy}, \neg\alpha_3 \\ \sim\mathit{stars} &\leftarrow \neg\alpha_4\} \end{aligned}$$

and $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$.

When the abductive program is interpreted, there are 16 possible combinations ($2^{|\mathcal{A}^*|}$)

to include the abducibles with the following GAS's in this case:

$$\mathcal{M}(\Delta_1) = \{\sim stars, \alpha_1, \alpha_2\}; \quad \mathcal{M}(\Delta_2) = \{day, \sim stars\};$$
$$\mathcal{M}(\Delta_3) = \{day, \sim stars, \alpha_2\}; \quad \mathcal{M}(\Delta_4) = \{day, \sim stars, \alpha_2, \alpha_3\};$$
$$\mathcal{M}(\Delta_5) = \{\sim stars, \alpha_1, \alpha_2, \alpha_3\}; \quad \mathcal{M}(\Delta_6) = \{day, \sim stars, \alpha_3\};$$
$$\mathcal{M}(\Delta_6) = \{night, \sim stars, \alpha_3\}; \quad \mathcal{M}(\Delta_7) = \{night, \sim stars, \alpha_1, \alpha_3\};$$
$$\mathcal{M}(\Delta_9) = \{day, \alpha_2, \alpha_4\}; \quad \mathcal{M}(\Delta_{10}) = \{day, \alpha_2, \alpha_3, \alpha_4\};$$
$$\mathcal{M}(\Delta_{11}) = \{, \alpha_1, \alpha_2, \alpha_4\}; \quad \mathcal{M}(\Delta_{12}) = \{, \alpha_1, \alpha_2, \alpha_3, \alpha_4\};$$
$$\mathcal{M}(\Delta_{13}) = \{day, \alpha_3, \alpha_4\}; \quad \mathcal{M}(\Delta_{13}) = \{night, \alpha_3, \alpha_4\};$$
$$\mathcal{M}(\Delta_{14}) = \{night, \alpha_1, \alpha_3, \alpha_4\}; \quad \mathcal{M}(\Delta_{16}) = \{night, stars, constls, \alpha_4\};$$
$$\mathcal{M}(\Delta_{16}) = \{day, \alpha_4\}; \quad \mathcal{M}(\Delta_{17}) = \{night, stars, constls, \alpha_1, \alpha_4\}$$

where

$$\Delta_1 = \{\alpha_1, \alpha_2\}; \quad \Delta_2 = \{\};$$
$$\Delta_3 = \{\alpha_2\}; \quad \Delta_4 = \{\alpha_2, \alpha_3\};$$
$$\Delta_5 = \{\alpha_1, \alpha_2, \alpha_3\}; \quad \Delta_6 = \{\alpha_3\};$$
$$\Delta_7 = \{\alpha_1, \alpha_3\}; \quad \Delta_9 = \{\alpha_2, \alpha_4\};$$
$$\Delta_{10} = \{\alpha_2, \alpha_3, \alpha_4\}; \quad \Delta_{11} = \{\alpha_1, \alpha_2, \alpha_4\};$$
$$\Delta_{12} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}; \quad \Delta_{13} = \{\alpha_3, \alpha_4\};$$
$$\Delta_{14} = \{\alpha_1, \alpha_3, \alpha_4\}; \quad \Delta_{16} = \{\alpha_4\};$$
$$\Delta_{17} = \{\alpha_1, \alpha_4\};$$

Next, the corresponding weights for each w are

$$w(1) = 1; w(2) = 5; w(3) = 21; w(4) = 85$$

and the sums of the corresponding weights:

$$s(\Delta_1) = 2; \quad s(\Delta_2) = 0; \quad s(\Delta_3) = 1;$$
$$s(\Delta_4) = 2; \quad s(\Delta_5) = 3; \quad s(\Delta_6) = 1;$$
$$s(\Delta_7) = 2; \quad s(\Delta_9) = 2; \quad s(\Delta_{10}) = 3;$$
$$s(\Delta_{11}) = 3; \quad s(\Delta_{12}) = 4; \quad s(\Delta_{13}) = 2;$$
$$s(\Delta_{14}) = 3; \quad s(\Delta_{16}) = 1; \quad s(\Delta_{17}) = 2$$

As a result, from the complete list of GAS*, it is easy to realise that its unique* MSGAS *is* $\{\sim stars, day\}$*, and its* ***update answer set*** $\mathcal{S} = \{\sim stars, day\}$*, as one would expect.*

This section consists of the main definitions of an *update semantics for sequences of programs* and they have been illustrated with a series of examples. The following natural step to understand this approach is by means of its main properties that characterise it as a good candidate to represent *dynamic knowledge*.

7.3 ⊗-Properties

This section shows the main results of the semantics for update sequences that satisfies basic structural properties introduced in Sections 6.2 and 6.2, as well as postulates of *Weak Irrelevance of Syntax* and *Strong Consistency*, just as ⊘-operator in Section 6.1. Notwithstanding, this current ⊗-operator can already deal with *multiple updates in a sequence*, as shown in Theorem 8.1 below.

As mentioned along the thesis, this set of properties can overcome the sort of problems introduced with Example 7.1 of having extra models when updating with either *redundant information* or *tautological rules*. As a consequence, any semantics of logic programs aimed to be generally accepted should meet at least the basic set of properties introduced in Section 6.2 and summarised below.

Accordingly, one can formulate the following theorem with the properties interpreted for and extended to sequences of updates.

7.3.1 Inconsistencies

Before proving theorem 7.1 for the called structural properties, it is necessary to introduce some preliminary results of this work that have to do with *inconsistencies*. Firstly, any consistent update to a knowledge base guaranties a consistent result. Formally,

Lemma 7.1 (⊗-weak consistency view)**.** *Suppose* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *are* ELP*'s and an update sequence* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ *with its corresponding abductive program* $\Pi_{\mathcal{A}^*} = \langle \Pi' \cup \Pi_n, \mathcal{A}^* \rangle$*. If* Π_n *is consistent then* $\Pi_{\mathcal{A}^*}$ *is consistent.*

sketch. Suppose $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are ELP's and Π_n consistent. This means that $\langle \Pi' \cup \Pi_n, \mathcal{A}^* \rangle$ has a generalised answer set $\mathcal{M}(\Delta)$ out of answer sets of $\Pi' \cup \Pi_n \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$, which is clearly consistent. Therefore, if Π_n is consistent, $\Pi_{\mathcal{A}^*}$ is also consistent. □

Corollary 7.2 (⊗-consistency Preservation)**.** *Suppose that* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *are* ELP*'s. The update sequence* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ *is consistent if* Π_n *is consistent.*

Corollary 7.2 also proves to be useful to *restore consistency* from an originally *inconsistent knowledge base*. This property is also known in the literature as *Consistency*

Preservation and is a general case of Sakama and Inoue's *inconsistency removal*. Note that the latter's sole name confirms the *syntactical orientation* of their approach. Moreover, the latter requires that $\Pi_0 \otimes \Pi_1 \subseteq \Pi_0$, which isn't applicable to $\otimes$-operator.

As a result, the following proposition follows directly from Corollary 7.2.

Proposition 7.2 (consistency restoration)**.** *Suppose* Π_0 *is an ELP's. The update* $\Pi_0 \otimes \emptyset$ *is consistent.*

Finally, the only reason to have an *inconsistency* in an $\otimes$-update sequence is by having an *inconsistent update*:

Proposition 7.3. *Suppose* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *are ELP's. If* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ *has no update answer sets, then* Π_n *has no answer sets.*

This preliminary results make a solid framework to introduce the main theorem of this section: meeting the set of structural properties studied along this research.

7.3.2 Structural Properties

Before listing these properties, it is necessary to recall from Section 3.3 that the statement $\Pi_1 \equiv \Pi_2$ that means that both Π_1 and Π_2 have the same answer sets —or alternatively $\Pi_1 \equiv_{\mathsf{ASP}} \Pi_2$. With an abuse of notation, when stating equivalence between update sequences, indeed it means that they have the same (or different) *Update Answer Sets*. Last but not least, notice that *equivalence between update sequences* of more than two programs seems to make sense at the level of weak equivalence rather than strong, and that is because each update sequence means one and only one update. As a consequence, there is no resulting knowledge base that might contain (in)complete information. Finally, relevant background like Theorem 3.2 from Section 2.1.3, is also necessary.

Accordingly, the set of *structural properties to update sequences of ELP's* can be listed as follows.

$\otimes$-SP-1, Addition of Tautologies [Eiter et al., 2002]: If Π has only *tautological clauses* of the form $\ell \leftarrow \ell$, any *consistent sequence*

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

$\otimes$-SP-2, Initialisation [Eiter et al., 2002]:

$$\emptyset \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

$\otimes$-SP-3, Inertia: If Π_n is consistent,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \emptyset \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

⊗-SP-4, Idempotence: For any sequence $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ with update answer sets,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

⊗-SP-6, Non-interference: If $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are programs defined over mutually *disjoint alphabets*, then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$$

⊗-SP-7, Augmented Update [Eiter et al., 2002]: If $\Pi_x \subseteq \Pi_y$ and Π_y is consistent, then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \otimes \Pi_y \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$$

⊗-SP-8, Strong Consistency, SC : If $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$ with $n \geq 2$ is consistent, then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$$

⊗-SP-8a, Weak Consistency, WC : If $\Pi_n \cup \Pi_m$ is consistent,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \cup \Pi_m \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_m$$

This is a particular case of property ⊗-SP-8, applicable to sequences of updates with $n > 2$.

⊗-SP-9, Weak Irrelevance of Syntax, WIS: If $T_{\mathcal{N}_2}(\Pi_x) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_y)$ then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$$

This is the set of minimal structured properties, fully introduced in Section 6.2, that an update semantics should meet in order to avoid counterintuitive results like the ones in Example 1.1 and recappedin Example 7.4. With this collection one can formulate the following theorem.

Theorem 7.1. *Given a ⊗-sequence of update extended logic programs $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$, with $n \geq 2$, over a set of atoms $\mathcal{A}$, ⊗-operator satisfies properties ⊗-SP-1 to ⊗-SP-9.*

Proof for Theorem 7.1. **⊗-SP-2, Initialisation:** $\emptyset \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$.

$\emptyset \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ has the abductive program $\langle\, \emptyset \cup \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_{n-1} \cup \Pi_n, \mathcal{A}^* \,\rangle$, that is the same abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_{n-1} \cup \Pi_n, \mathcal{A}^* \,\rangle$ from $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ and both of them have the same GAS's, and the same MSGAS's and thus

the same $\otimes$-update answer sets. Hence, $\emptyset \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ as required.

$\otimes$-SP-3, Inertia: $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \emptyset \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$.

Assume Π_n consistent. Then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \emptyset$ has the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n \cup \emptyset, \mathcal{A}^* \,\rangle$, which is logically equivalent to the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n, \mathcal{A}^* \,\rangle$ with $\mathcal{M}(\Delta)$ as its MSGAS. This means that $\mathcal{M}(\Delta) \leq_{\mathcal{S}} \mathcal{M}(\Delta')$ and that $\mathcal{M}(\Delta)$ is an answer set of $\Pi'_1 \cup \Pi'_2 \cup \ldots \Pi'_n \cup \{\alpha \leftarrow \top\}$ where $\alpha \in \Delta$ and $\Delta \cap \mathfrak{L}_{\Pi'_n} = \emptyset$. Thus, $s(\Delta) \leq s(\Delta')$ and $w(x) \leq w(n)$ for $x < n$, which is clearly true. Then $\mathcal{M}(\Delta)$ is also a MSGAS from the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi_n, \mathcal{A}^* \,\rangle$ out of $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$. That means both update sequences have the same $\otimes$-update answer sets. Therefore, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \emptyset \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ as required.

$\otimes$-SP-8a, Weak Consistency: If $\Pi_n \cup \Pi_m$ is consistent,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \cup \Pi_m \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_m$$

Suppose $\Pi_n \cup \Pi_m$ is consistent. Then, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_m$ has the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n \cup \Pi_m, \mathcal{A}^* \,\rangle$ with $\mathcal{M}(\Delta)$ as its MSGAS. Since $\Pi_n \cup \Pi_m$ is consistent, $\mathcal{M}(\Delta) \leq_{\mathcal{S}} \mathcal{M}(\Delta')$ with $\Delta \cap \mathfrak{L}_{\Pi'_n} = \emptyset$, and $s(\Delta) \leq s(\Delta'); w(x) \leq w(n); x < n$. Thus, $\mathcal{M}(\Delta)$ is the same MSGAS of the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi_n \cup \Pi_m, \mathcal{A}^* \,\rangle$ out of $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \cup \Pi_m$. This means that both update sequences have the same $\otimes$-update answer sets. Therefore, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \cup \Pi_m \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_m$.

$\otimes$-SP-8, Strong Consistency: If $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$ with $n \geq 2$ is consistent, then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$.

Assume $\mathcal{M}$ is an answer set of $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$ and $n \geq 2$. Then, $\mathcal{M}$ must be the same model of $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$. As $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ has the abductive program $\langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle$, and an arbitrary answer set of $\Pi' \cup \Pi_n \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$ is $\mathcal{M}(\Delta)$, then the MSGAS's of $\langle \Pi' \cup \Pi_n, \mathcal{A}^* \rangle$ are just $\mathcal{M}(\emptyset)$. Because the MSGAS is $\mathcal{M}(\emptyset)$, then the literals in $\mathfrak{L}_{\langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle} \cap \mathcal{A}^*$ are not positive and Π' is an ordinary extended logic program whose semantics coincides with $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_{n-1}$. Therefore, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$, as required.

$\otimes$-SP-1, Addition of Tautologies [Eiter et al., 2002]: If Π has only *tautological*

rules of the form $\ell \leftarrow \ell$, any *consistent sequence*

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

Suppose $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ consistent. This means that $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi$ has the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n \cup \Pi, \mathcal{A}^* \,\rangle$ that is logically equivalent to $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n, \mathcal{A}^* \,\rangle$ with $\mathcal{M}(\Delta)$ as its MSGAS. Then by ⊗-SP-3, $\mathcal{M}(\Delta)$ are also the same MSGAS's than the ones from the abductive program $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi_n, \mathcal{A}^* \,\rangle$, which is the abductive program of $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$. This means both update sequences have the same ⊗-update answer sets. Therefore, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$, as required.

⊗-SP-4, Idempotence: For any sequence $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ with update answer sets,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

Suppose $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ has update answer sets. Then, the update sequence $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ has the ⊗-abductive program $\Pi_{\mathcal{A}^*} = \langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_{n-1} \cup \Pi'_n \cup \Pi''_1 \cup \Pi''_2 \cup \cdots \cup \Pi''_{n-1} \cup \Pi_n, \mathcal{A}^* \,\rangle$ where Π''_z is the relaxed program of Π_z such that $(\Pi''_z \cap \mathcal{A}^*) \cap (\Pi'_z \cap \mathcal{A}^*) = \emptyset$ and thus, the weakened rules of Π_z by Π'_z are the same weakened rules by Π''_z in every GAS of $\Pi_{\mathcal{A}^*}$. Hence, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$, as required. Therefore, for a sequence $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ with update answer sets,

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$$

⊗-SP-9, Weak Irrelevance of Syntax: If $T_{\mathcal{N}_2}(\Pi_x) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_y)$ then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y.$$

Suppose $T_{\mathcal{N}_2}(\Pi_x) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_y)$ Then, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x$ and $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$ have the respective abductive programs $\langle\, \Pi' \cup \Pi_x, \mathcal{A}^* \,\rangle$ and $\langle\, \Pi' \cup \Pi_y, \mathcal{A}^* \,\rangle$, whose respective GAS's come from programs $\Pi' \cup \Pi_x \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$ and $\Pi' \cup \Pi_y \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$ with $\Delta \subseteq \mathcal{A}^*$ and with the same set of abducibles. By Theorem 3.2, $\Pi' \cup \Pi_x \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\} \equiv \Pi' \cup \Pi_y \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$. Consequently, $\langle\, \Pi' \cup \Pi_x, \mathcal{A}^* \,\rangle$ and $\langle\, \Pi' \cup \Pi_y, \mathcal{A}^* \,\rangle$ have the same GAS's, the same MSGAS's, and $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$, as required.

Therefore, if $T_{\mathcal{N}_2}(\Pi_x) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_y)$, then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$.

⊗-SP-7, Augmented Update: If $\Pi_x \subseteq \Pi_y$ and Π_y is consistent, then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \otimes \Pi_y \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$.

Suppose Π_y is consistent and $\Pi_x \subseteq \Pi_y$. This implies that $\Pi_x \cup \Pi_y = \Pi_y$ and $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$ is logically equivalent to $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \cup \Pi_y$. Then, by weak consistency, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \otimes \Pi_y \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$, as required. Therefore, if $\Pi_x \subseteq \Pi_y$ and Π_y is consistent, then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_x \otimes \Pi_y \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \otimes \Pi_y$.

⊗-SP-6, Non-interference: If $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are programs defined over mutually *disjoint alphabets*, and either all of them are consistent or are not, then

$$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$$

Assume that $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are defined over disjoint alphabets and that all of them are consistent. Then, both $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n$ and $\Pi_n \cup \Pi_{n-1} \cup \cdots \cup \Pi_1$ are consistent too and by the commutative law on the union[1] , they may have any order. Thus, by Strong Consistency on any of the orders of the union, $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_n \equiv \Pi_n \cup \Pi_{n-1} \cup \cdots \cup \Pi_1 \equiv \Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$, as required.

Now suppose that $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ all are inconsistent. Then, the ⊗-update sequences $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ and $\Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$ have their respective abductive programs $\langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle$ and $\langle\, \Pi' \cup \Pi_1, \mathcal{A}^* \,\rangle$, which are clearly inconsistent and $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$, as required.

Therefore, if $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ are programs defined over disjoint alphabets, and either all of them are consistent or not, then $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n \equiv \Pi_n \otimes \Pi_{n-1} \otimes \cdots \otimes \Pi_1$.

□

This is the *minimal set of properties* considered as fundamental for a proper update semantics as explained in Section 6.2, and ⊗-operator satisfies at least this collection of properties.

One immediate result from Theorem 7.1 is the equivalence between an abductive program of the form $\Pi_{\mathcal{A}^*}$ in Definition 8.6 where Π_n is also α-relaxed and another where Π_n is not relaxed. This is formally expressed in Proposition 7.4 as follows.

[1]Notice that associativity law does not apply to ⊗-operator.

Proposition 7.4 (Full Relaxation)**.** *Suppose* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *are ELP's and* Π_n *consistent. The abductive program* $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n, \mathcal{A}^* \,\rangle$*, out of the* α*-relaxed programs from the sequence has the same MSGAS's as the abductive program* $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi_n, \mathcal{A}^* \,\rangle$ *does.*

Another nice property has to do with *updates to inconsistent programs* that shall be formalised in upcoming sections. Meanwhile, consider the following update that recovers consistency of a sequence with an inconsistency.

Example 7.5. *Suppose an update to* Π_1 *with* Π_2*, and with* Π_3*, where* $\Pi_1 = \{a \leftarrow \top\}$*,* $\Pi_2 = \{(b \leftarrow \top), (\sim b \leftarrow \top)\}$ *and* $\Pi_3 = \{\}$*. Its abductive program corresponds to* $\langle\, \Pi', \mathcal{A}^* \,\rangle$*, where*

$$\begin{aligned} \Pi' = \{a &\leftarrow \neg\alpha_1 \\ b &\leftarrow \neg\alpha_2 \\ \sim b &\leftarrow \neg\alpha_3\} \end{aligned}$$

and $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3\}$*, whose GAS's are*

$$\begin{aligned} \mathcal{M}(\Delta_1) = \{a, \sim b\}_{\{\alpha_2\}}; &\qquad \mathcal{M}(\Delta_2) = \{\sim b\}_{\{\alpha_1,\alpha_2\}}; \\ \mathcal{M}(\Delta_3) = \{a, b\}_{\{\alpha_3\}}; &\qquad \mathcal{M}(\Delta_4) = \{b\}_{\{\alpha_1,\alpha_3\}}; \\ \mathcal{M}(\Delta_5) = \{a\}_{\{\alpha_2,\alpha_3\}}; &\qquad \mathcal{M}(\Delta_6) = \{\}_{\{\alpha_1,\alpha_2,\alpha_3\}} \end{aligned}$$

$$\begin{aligned} \Delta_1\{\alpha_2\}; &\qquad \Delta_2\{\alpha_1, \alpha_2\}; \\ \Delta_3\{\alpha_3\}; &\qquad \Delta_4\{\alpha_1, \alpha_3\}; \\ \Delta_5\{\alpha_2, \alpha_3\}; &\qquad \Delta_6\{\alpha_1, \alpha_2, \alpha_3\} \end{aligned}$$

and the corresponding weights for each w *are*

$$w(1) = 1; w(2) = 4.$$

Next, the corresponding sum of weights for each Δ *are*

$$\begin{aligned} s(\Delta_1) = 4; &\qquad s(\Delta_2) = 5; \\ s(\Delta_3) = 4; &\qquad s(\Delta_4) = 5; \\ s(\Delta_5) = 8; &\qquad s(\Delta_6) = 9 \end{aligned}$$

that lead to two MSGAS's, $\{a, \sim b\}_{\{\alpha_2\}}, \{a, b\}_{\{\alpha_3\}}$ *from which one can conclude that the*

update answer sets are just $\{a, b\}, \{a, \sim b\}$.

This example shows how to prevent a knowledge base from *collapse due to an inconsistency*. One solution, as Example 7.5 shows, is introducing a *disjunction* until new evidence supports either conclusion. Another solution, as in the following section, is *relaxing the updating program* as well, which produces the same effect as in the example above.

7.4 $\otimes'$-Operation

One may find more interesting results when analysing Proposition 7.4 and defining a slight different construction to update sequences of EDLP's. Such a difference consists in relaxing not only the previous sequence to the update program, but also the update program itself! That shall produce the same effect as in Example 7.5 above, as well as interesting properties like updating with an inconsistency, formalised in Section 7.5 below, and compared in Appendix A.

Definition 7.3. *Given an update sequence of ELP's,* $\Pi_1 \otimes' \Pi_2 \otimes' \cdots \otimes' \Pi_n$*, over a set of atoms* $\mathcal{A}$ *with* n *as a natural number; with its corresponding* α*-relaxed sequence* $(\Pi'_1, \Pi'_2, \ldots, \Pi'_n)$*, where* $\alpha \in \mathcal{A}^*$*; and the abductive program* $\Pi_{\mathcal{A}^*} = \langle \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n, \mathcal{A}^* \rangle$ *with* $\mathcal{M}(\Delta)$ *as the GAS's of* $\Pi_{\mathcal{A}^*}$*; and* $w(l); l; s(\Delta)$ *defined as in Definition 8.6:*

- $\mathcal{M}(\Delta_1) \leq_{\mathcal{S}} \mathcal{M}(\Delta_2)$ *if and only if* $s(\Delta_1) \leq s(\Delta_2)$.
- $\mathcal{M}(\Delta_1) \equiv_{\mathcal{S}} \mathcal{M}(\Delta_2)$ *if and only if* $s(\Delta_1) = s(\Delta_2)$.
- $\mathcal{M}(\Delta)$ *is an MSGAS of* $\Pi_{\mathcal{A}^*}$ *if and only if* $\mathcal{M}(\Delta)$ *is minimal with respect to* $\leq_{\mathcal{S}}$.

Finally, *Update Answer Sets* are the models one expects from an updating sequence:

Definition 7.4 ($\otimes'$-update Answer Set)**.** *Given a* $\otimes'$*-sequence of updating ELP's,* $\Pi_1 \otimes' \Pi_2 \otimes' \cdots \otimes' \Pi_n$*, with* $n \geq 2$*, over a set of atoms* $\mathcal{A}$*, the set* $\mathcal{S} \subseteq \mathcal{A}$ *is its* update answer set *if and only if* $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ *for some* minimal sequenced generalised answer Set $\mathcal{S}'$ *of the sequence, and* $\otimes'$ *is the corresponding* update operator.

Besides the properties shared with $\otimes$-operator, the following section includes an interesting property of *consistency view* for $\otimes'$-operator.

7.5 $\otimes'$-Properties

Having introduced Proposition 7.4 and $\otimes'$-operator, the following results illustrate another alternative to *preserving a knowledge base from collapse*, due to an *inconsistency update*.

Proposition 7.5 (Full Relaxation Consistency)**.** *Suppose* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *are ELP's and* Π'_x *as the corresponding* α*-relaxed program with* $1 \leq x \leq n$*. The abductive program* $\langle\, \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_n, \mathcal{A}^* \,\rangle$ *is consistent.*

This proposition also yields the following obvious result:

Corollary 7.3 (Strong Consistency View)**.** *Suppose* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *are ELP's. The update sequence* $\Pi_1 \otimes' \Pi_2 \otimes' \cdots \otimes' \Pi_n$ *is consistent.*

Another interesting result is its equivalence with the operator just introduced in Section 7.2, which shows common properties.

Proposition 7.6. *Given the sequence* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *of EDLP's with* Π_n *consistent,* $\Pi_1 \otimes' \Pi_2 \otimes' \cdots \otimes' \Pi_n \equiv \Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$

Proof. This is a straightforward consequence of Proposition 7.4 and Definition 7.3. □

Corollary 7.4 ($\otimes'$-structural properties)**.** $\otimes'$ *operator satisfies properties* ⊗*-SP-1–*⊗*-SP-9.*

This section has introduced a more general update operator that, besides satisfying the same properties than ⊗-operation, it is also more *robust* when new information that is inconsistent arises.

Notice that Lemma 7.1, together with the resulting corollary and Proposition 7.3 are properties that suggest a classification of this approach as a framework for *belief change* rather than *belief updates*.[1] One of the main goals of this work, however, is *representation of knowledge* in general by producing a framework with an *intuitive behaviour*, even though the title of this chapter includes the word "updates" for historical and practical reasons. Another difference from Katsuno and Mendelzon is whether the knowledge base is evolving in a *changing environment* or not, which would come to a subjective statement. As a result, no strict distinction shall be made in this semantics with respect to *belief revision* and *belief update*. It is also worth noticing that Sakama and Inoue have a similar position with respect to such a difference in update semantics, where they suggest ambiguous results may arise when trying to state a clear difference under some contexts [Sakama and Inoue, 2003].

Finally, the following sections present an implemented prototype to automatically update sequences of logic programs in ASP for the semantics in Section 7.2. This sort of programs for ASP problem solving are known as *solvers* in the literature.

7.6 ⊗-prototype

One of the latest ideas introduced in Chapter 6 is choosing between generalised answer sets to overcome disadvantages of previous approaches described in Chapter 4.

[1] Further details on such differences may be found in Section 2.4.

However, a solver to automatically compute this semantics is necessary for different demanding proposes that go from a classroom tool and assistant, to implementations of more complex prototypes aimed at exploiting knowledge-management fields and further properties discovery.

7.6.1 Implementing Updates on DLV

As already presented in Section 3.1, there are two major efficient *solvers* to compute ASP with a vast background of implementation and research. They are DLV and SMODELS and the system proposed in this section employs the former at a higher abstraction level in order to *update ELP programs*. Towards this end, this section is an introduction to a transformation that may be interpreted in either system with some slight syntactic adaptations[1].

To begin with, an approach to implement an update semantics in MGAS has already been introduced in Chapter 6 by means of *preferred disjunctive logic programs* in Brewka's ODLP and has a *solver for pairs of programs* at `http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html`, as described in Section 6.3. However, the mentioned update semantics and thus the final system itself, are limited to *single updates*.

Indeed, a justification to use ODLP is that there is a solver available named PSmodels[2] that is an extension to SMODELS to compute *preferred answer sets*. Unfortunately, up to the printout of this chapter, there is no reliable version of PSmodels and the latest one (v.2.26a) endures some few *bugs* under certain circumstances[3]. In addition to the *running solver*, it is believed that DLV significantly outperforms SMODELS, not to mention that ODLP is such a colossal semantics that can do much more complex tasks than just computing MGAS's, which might compromise the performance of the desired system and mix up its simple formulation.

As a result this section is an introduction to the use of DLV's *Weak Constraints* for their characteristics of *optimisation in preferences* [Leone et al., 2006] that enjoys the above benefits of being in DLV with no more extra throughput added to the system.

This section consists of an implementation of the declarative version for updates sequences in Section 7.2, which is proposed as an alternative to other *syntax-based semantics* described in Chapter 4. One of the main contributions from this implementation is to use DLV's *Weak Constraints* to compute the model(s) of any update sequence, besides providing an *online solver for public experiments*.

The structure of this section consists of the basic configuration of the system, a description of the employed technology, a discussion of the major process to compute the intended models, as well as a set of examples to illustrate its use. Before going straight to the system description, a short recapitulation of its background is necessary to understand its foundation on ASP, already introduced in Section 3.1,abductive programming from Section 3.6, and in particular DLV's *weak constraints*.

[1] Refer, for instance, to [Leone et al., 2006] for an *equivalence of weak constraints* in SMODELS.

[2] `http://www.tcs.hut.fi/Software/smodels/priority/`

[3] Try to compute the preferred models of a simple program like $\{a.\}$.

7.6.2 DLV's Weak Constraints

Let us briefly recap some few formulas for weak constraints, which are fully introduced in Section 3.4.

Leone et al. introduced a nice feature of DLV solver known as *Weak Constraints* that may be employed to set up preferences between models. In particular, a *weak constraint* is a variant of an *integrity constraint* that may be violated in order to establish priorities amongst models. One of its differences is the introduction of a new derivation symbol ":∼", rather than ":– " or "←". Moreover, one can specify the priority level and weight of the constraint. Formally,

Definition 7.5 (Weak Constraint [Leone et al., 2006])**.** *A* weak constraint *(ω) is an expression of the form*

$$:\sim \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m[w : p] \tag{7.5}$$

where for $0 \leq k \leq m$*,* $\ell_1, \ldots, \ell_m$ *are literals, while* w *(the weight) and* p *(the level, or layer) are positive integer constants or variables.*

In addition, $\Omega(\Pi)$ shall denote the finite *set of weak constraints* occurring in a given program Π. Likewise, a *ω-program* is a logic program with weak constraints.

In order to provide a more syntactic sugar, another way to define a weak-consitraint expression from Definition 8.10 is as follows.

Definition 7.6 (Weak Constraint)**.** *A* weak constraint *(ω) is an expression of the form*

$$[w : p] \leftarrow \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m \tag{7.6}$$

where for $0 \leq k \leq m$*,* $\ell_1, \ldots, \ell_m$ *are literals, while* w *(the weight) and* p *(the level, or layer) are positive integer constants or variables.*

From now on, the previous weak-constraints form shall be employed in the context of DLV-code, while the other in higher abstraction levels.

Similarly to integrity constraints in Section 3.1, one may say that a weak constraint $\rho = ([w : p] \leftarrow \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m)$ is *violated* by an answer set $\mathcal{S}$ of a program Π if the following three conditions hold:

1. $\rho \in \Pi$
2. $\{\ell_1, \ldots, \ell_k\} \subseteq \mathcal{S}$
3. $\{\ell_{k+1}, \ldots, \ell_m\} \nsubseteq \mathcal{S}$

Additionally, Leone et al. simplify the combination of weights in levels by introducing a function $H^{\Pi}(\mathcal{S})$ that grows in direct proportion to the weight and level of the weak constraint as follows:

Definition 7.7 (Objective Function, $H^{\Pi}(\mathcal{S})$ [Leone et al., 2006])**.** *Given a ground program* Π *with weak constraints* $\Omega(\Pi)$ *and an answer set* $\mathcal{S}$*, the* ω objective function $H^{\Pi}(\mathcal{S})$ *is defined by using an auxiliary function* f_{Π} *that maps levelled weights to weights without levels:*

$$\begin{aligned} f_{\Pi}(1) &= 1 \\ f_{\Pi}(n) &= f_{\Pi}(n-1) \cdot |\Omega(\Pi)| \cdot w^{\Pi}_{max} + 1, n > 1 \\ H^{\Pi}(\mathcal{S}) &= \sum_{i=1}^{l^{\Pi}_{max}} (f_{\Pi}(i) \cdot \sum_{\rho \in N^{\Pi}_{i}(\mathcal{S})} weight(\rho)) \end{aligned}$$

where $N^{\Pi}_{i}(\mathcal{S})$ *denotes the weak constraints at level* i *violated by* $\mathcal{S}$*, and* $weight(\rho)$ *the weight of weak constraint* ρ*.*

Finally, the *best models* of such a logic program are those that *minimise* the number of *violated weak constraints.*

Definition 7.8 (Weak-Constraint Model [Leone et al., 2006])**.** *For an EDLP* Π *with weak constraints, a set* $\mathcal{S}$ *is a* weak-constraint model *of* Π *if and only if*

1. $\mathcal{S}$ *is an answer set of* Π
2. $H^{\Pi}(\mathcal{S})$ *is minimal over all the answer sets of* Π*.*

When the underlying semantics is ASP in Definition 8.13, a weak-constraint model is also known as *Optimal Answer Set.*

Moreover, the language of EDLP's with weak constraints shall be called DATALOG$^{\vee,\omega}$, which is very similar to the notation from the literature.

Further details on the characterisation of the transformation shall be provided later in Section 8.4.2, with an equivalent simpler operator.

7.6.3 The Parser

Differently from the implementation proposed in Section 6.3.6, which has a parser embedded in its `PHP`[1] code, I have implemented the new parser presented in this section, it is compiled in `C` on the MacOS X^{TM} platform at Darwin^{TM} level, as well as for a Linux alternative at the same location: `http://www.in.tu-clausthal.de/~guadarrama/updates/seqs.html`. The advantage of having a *UNIX binary module* is the ease to be plugged in to other modules so as to form more complex applications.

On the other hand, MacOS X^{TM}/Darwin^{TM} is a BSD branch of UNIX, that has a ported set of ***Lex*** and ***Yacc*** utilities in their GNU versions of ***Flex*** and ***Bison*** — respectively. `Flex` is a short name for *Fast Lexical Analyser* that generates code to scan text through *regular-expressions pattern matching.*

[1] This is a script language quite suitable for small processes of dynamic contents on web pages.

In particular, the following specification shows the main tokens implemented for this update solver.

```
NAME        [-~]?[[:alnum:]_]+
PnSTART     "{"
PnEND       "}"
GETS        ":- "
NOT         "not "
RULEEND     "."
CONJUNCT    ","
DISJUNCT    "|"
```

Tokens like PnSTART and PnEND split the sequence of programs into individual programs, while the rest of the tokens need no explanation.

As soon as `Flex` decomposes the text of a program sequence, its output is taken on by `Yacc`, which gives a meaning to each correct structure of rules and program sequence. In particular, the `Yacc` process specifies a *grammar for update sequences*. It is also responsible for weakening each rule in all programs of the sequence with a new unique atom and establishes a *cardinality-preference relation* amongst such atoms, according to the sequence the rule is in. Last, this process is responsible of an *error-checking mechanism* that verifies the correctness of a given program sequence according to the BNF grammar below.

```
<sequence>    ::=<sequence> '{' <program> '}'
                |
<program>     ::=<program> <rule>
<rule>        ::=<head> END
                | <head> GETS <body> END
<head>        ::=<POST>
                |
<POST>        ::=<LITERAL>
                | <POST> DISJUNCT <LITERAL>
<body>        ::=<PRE>
                |
<PRE>         ::=<LITERAL>
                | NOT <LITERAL>
                | <PRE> CONJUNCT <LITERAL>
                | <PRE> CONJUNCT NOT <LITERAL>
<LITERAL>     ::=NAME
                | NAME '(' NAME ',' NAME ')'
```

As mentioned before, for each rule that is analysed, the system appends a pair of new rules with *weakening atoms* and a weak constraint, in programs previous to the

last one in the sequence, as follows:

$$[1:p] \leftarrow \alpha_i \tag{7.7}$$
$$\sim\alpha_i \mid \alpha_i \leftarrow \top \tag{7.8}$$

where i represents the i^{th} abducible α and p the p^{th} program, the latter forming a *weight-level relation* that corresponds to $[1:p]$.

The intuition behind this formulation is computing the MSGAS of the abductive program by violating the least number of weak constraints. In addition, the $[1:p]$ relation represents the sequence order from Definition 8.6. That is to say, the models are those that have the least weight-level relation.

Example 7.6. *Suppose the sequence*

$$\begin{aligned}\Pi_1 &: \quad \{a \leftarrow \neg b\}\\ \Pi_2 &: \quad \{(b \leftarrow \neg a); (c \leftarrow a); (d \leftarrow \sim a, b)\}\\ \Pi_3 &: \quad \{e \leftarrow \neg b\}\end{aligned}$$

The corresponding abductive program is $\langle\, \Pi' \cup \Pi_3, \mathcal{A}^* \,\rangle$ *where*

$$\begin{aligned}\Pi' = \{&a \leftarrow \neg b, \neg\alpha_1 \qquad b \leftarrow \neg a, \neg\alpha_2\\ &c \leftarrow a, \neg\alpha_3 \qquad d \leftarrow \sim a, b, \neg\alpha_4 \qquad e \leftarrow \neg b\}\end{aligned}$$

and $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. *Such an abductive program is transformed into a* ω*-program in* DLV *as*

$$\begin{array}{lll} a \leftarrow \neg b, \neg\alpha_1 & \sim\alpha_1|\alpha_1 \leftarrow \top & [1:1] \leftarrow \alpha_1\\ b \leftarrow \neg a, \neg\alpha_2 & \sim\alpha_2|\alpha_2 \leftarrow \top & [1:2] \leftarrow \alpha_2\\ c \leftarrow a, \neg\alpha_3 & \sim\alpha_3|\alpha_3 \leftarrow \top & [1:2] \leftarrow \alpha_3\\ d \leftarrow \sim a, b, \neg\alpha_4 & \sim\alpha_4|\alpha_4 \leftarrow \top & [1:2] \leftarrow \alpha_4\\ e \leftarrow \neg b, \neg\alpha_5 & \sim\alpha_5|\alpha_5 \leftarrow \top & [1:3] \leftarrow \alpha_5 \end{array}$$

This is the final stage before interpreting the *preferred transformed program* in DLV with weak constraints, as explained below.

7.6.4 The Top Module

The *top module* consists of a display of the original sequence, its transformation to abductive program, as well as the result of interpreting such an abductive program under MSGAS and the update answer sets of the sequence. All in all is coded into a

UNIX script, with some simple sub-processes that filter in the needed text from the formatted output in DLV. Last, this main module is also responsible of dealing with the *user interface* in HTML by getting the user's input sequence into a *text pane* of a web page and processing it to display the output within a new web page.

In a black-box system approach, the main module consists of an HTML page with a text panel to capture the user input of a sequence of ELP, with each program enclosed in braces "{}". In the same page, there is another text pane to capture switches for DLV. Once the user pushes the process button, the system processes the input text and yields an output divided into the original input sequence; its corresponding abductive program; the GAS's of the abductive program and the corresponding update answer set(s) of the sequence, when they exist.

7.6.4.1 The Abductive Program

The *abductive program* is, indeed, coded into a *cardinality-preference relation* of weak constraints. The relation consists of a *weakened program*, where each rule has its corresponding *pair of disjunctive abducibles* (7.8), and a weak constraint (7.7).

This simple process constructs a *triple rule* at the *parser stage* by keeping a counter for each abducible and one for each program in the sequence, which is displayed once a rule is recognised: the *relaxed rule*, a *disjunctive rule* and a *leveled weak constraint*.

7.6.4.2 Computing MSGAS's

Computing MSGAS's is a straightforward process that takes the *abductive preference program* from the previous process as an input and hands it to DLV. The general intuition behind this *optimisation solver* is computing the ASP reduct (Definition 3.4) of the ELP input program that returns none or more answer sets. Then, it chooses the *best weak-constraint model*(s). As mentioned before, the best answer set(s) are those that violate the least number of weak constraints (they all have the same weight) at the highest level.

From Example 7.6, the system returns the following two MSGAS's:

$$\{b\}_{\{-\alpha_1,-\alpha_2,-\alpha_3,-\alpha_4,-\alpha_5\}}, \{a,c,e\}_{\{-\alpha_1,-\alpha_2,-\alpha_3,-\alpha_4,-\alpha_5\}}$$

7.6.4.3 The Update Answer Sets

Finally, the last stage is a simple filtering with UNIX processes of the abductive atoms that omits them and gives the desired result. For this example, just $\{b\}, \{a,c,e\}$.

7.6.5 ⊗-Complexity

Now it is time to say a few words about *complexity of the ⊗-operator*. One may divide the main problem of updating a sequence $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ into two basic processes. Firstly, there is an implicit simple algorithm in the definition of an α-relaxed program

that transforms an ELP into a ω-program. Secondly, the resulting ω-program should be computed by DLV in order to check its models, which are called *optimal answer sets*.

By following the notation introduced in Section 3.7, the complexity of a DLV program with weak constraints is known to be exponential, bounded by co-NEXPTIME$^{\text{NP}}$ [Leone et al., 2006]. The *program transformation*, on the other hand, may be reduced to the problem of *tagging* each rule of the programs in the sequence $(\Pi_1, \Pi_2, \ldots, \Pi_{n-1})$ with $1, 2, \ldots, r$ in the definition of α-relaxed program, where $r = \sum_{i=1}^{n-1} |\Pi_i|$. As a consequence, the following obvious result holds.

Proposition 7.7. *The complexity of transforming any sequence* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *of grounded ELP's into an* α*-relaxed program is linear with respect to* $r = \sum_{i=1}^{n-1} |\Pi_i|$.

Finally, the size of the input to DLV, produced by weakening rules and introducing abducibles, grows *exponentially in the number of rules* in $\Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_{n-1}$ by having an upper bound of $2^{|\mathcal{A}^*|}$ possible *combinations to include abducibles* into every interpretation —see Example 7.4.

As a result, computing weak constraints and any grounded program DLP's is very expensive for known to be at the *second level of the polynomial hierarchy* [Eiter et al., 2002]. Given the exponential input size of a grounded relaxed sequence, however, the overall complexity lies in the exponential hierarchy with respect to the original non-relaxed sequence:

Proposition 7.8. *The complexity of computing a* $\otimes$*-update answer set of any sequence* $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ *of grounded ELP's is* exponential *with respect to* $r = \sum_{i=1}^{n-1} |\Pi_i|$, *bounded by co-NEXPTIME*^{NP}.

This is a clear shortcoming for semantics of updates of logic programs in ASP so as to have typical *industrial applications* of, say, *updating large knowledge bases*. Notwithstanding, the implementation to process *toy examples* in the classroom is an immediate practical application, as well as others like a *properties testbed* and more complex prototypes, to mention a few, that shouldn't be despised.

7.6.6 Discussion

This section has introduced general methods for *rapid prototyping* in DLV of logic programming semantics and for further research in *optimisation techniques*, as well as directions to implement the declarative version of both an update semantics and MSGAS's. The system has been developed with strong emphasis on *declarative programming*, in just some few fragments of imperative-procedural modules, in order to make it easily modifiable for particular frameworks and to confirm claims of the original semantics here presented. Another of its highlights is its modularity and *UNIX paradigm* that allows it to be a *web service* and easily plugged into other systems via on-line even without needing to download it. Moreover, its simple standard *graphical user interface* in HTML makes it very easy to use, compared to most of the solvers implemented for command-line use.

As one of the main components of Logic Programming, implementation of semantics helps quickly understand it (for educational proposes and for a reliable comparison tool, for instance), spread it, test properties and compute knowledge bases for more complex prototypes and other frameworks. In addition, an analysis on the *complexity* of the prototype shows that the transformation from a given sequence of logic programs is polynomial, while computing the resulting transformed program may be exponential, due to DLV's weak constraints. Nevertheless, the implementation shouldn't be despised, as it has immediate practical academic applications at least.

7.7 Conclusions for Chapter 7

This chapter is a proposal of a semantics for updates that performs the methods presented in Chapter 6 of relaxing an original knowledge base and of establishing preferences amongst candidate models. In particular, the intuition behind the methodology consists in favouring the latest updates that conflict with an original knowledge base by means of *preferring generalised answer sets*, and also by preserving *consistency*. In addition, it emphasises the importance of an approach based on key *structural properties*, and it may be used for EDLP's.

The core of this chapter shows that the new semantics satisfies the introduced set of *structural properties for sequences* of updating logic programs, as well as other needed properties of *Consistency Preservation* and *inconsistent updates*. The latter set of properties has to do with *Consistency Restoration* both from an original inconsistent knowledge base and an inconsistent observation, which makes the *difference between belief revision and updates* a little fuzzy. Nevertheless, such a difference is secondary in this research, as the general goal is to produce a robust semantics to represent knowledge, with intuitive behaviour.

As a result, the chapter consists of two alternatives to restore consistency. Firstly, a null update to an inconsistent knowledge base shall relax the source of inconsistency. Secondly, a slight modification to the semantics so that it relaxes the entire sequence including the update.

Besides satisfying the principles proposed, this chapter illustrates through several examples and transformations how to overcome problems occurring in alternative update approaches for ASP. One of the key transformations yields an abductive program out of a given sequence of updating extended logic programs. Another transformation constructs a *preferred weak-constraint program* to find *generalised answer sets* of the abductive program. Finally, as one main goal of Logic Programming, Section 7.6 includes a *functional prototype* from the declarative semantics version, that finds the update answer sets of a given sequence by means of DLV's *weak-constraints models*. The prototype is fully functional and runs online with a standard browser interface. The section also includes an analysis of complexity of the corresponding processes.

Chapter 8

Generalised Update

One of the goals of *commonsense reasoning* is to make an agent intelligent that may be autonomous. As suggested in the logic-programming literature, such a goal requires a solid theoretical basis on knowledge representation and nonmonotonic reasoning, and in particular, in *knowledge updates*. When dealing with knowledge updates, one needs a way to avoid inconsistencies due to *potential contradictory information* upcoming from *new evidence*. Much work has been done in the context of logic programming based on a common Answer Set Programming basis (hereafter ASP) by satisfying certain properties and postulates: [see Alferes et al., 2005; Eiter et al., 2002; Osorio and Zacarías, 2004; Sakama and Inoue, 2003; Zhang and Foo, 2005, e.g.]. However, despite the existence of several semantics for updates and a deep analysis of general principles, we are still far from having a general one that can satisfy many existing and well-known principles to represent *"correct" dynamic knowledge*.

In particular, Sakama and Inoue introduce and interesting *persistence situation* that the others do not manage well for different reasons, mainly for their approach of *sequence of updates* —see also Section 4.3. However, their semantics is strongly based on syntactical changes and it lacks of a proper characterisation of principles for updates, due to their different goals in their approach.

With the aim to define a general semantics and to succeed in the mentioned persistence situation, this chapter consists of another approach to the proposals in previous chapters, founded on GAS too, and based upon further principles for belief change that make it *independent of syntax*.

Besides satisfying most of belief revision postulates, this chapter exhibits an approach that consists in performing *successive updates* so that the semantics can deal with the problem that, according to Sakama and Inoue, produces *counterintuitive interpretations* in most current approaches. In addition to that, it meets at least the *structural properties* presented in Section 6.2., as well as five of the six most representative *postulates for belief change* in the literature.

In general, the main contributions of this chapter can be summarised as follows:

- a characterisation of five out of six AGM-postulates.

- a study of *Consistency Preservation* and *Consistency Restoration* within a framework for updates of logic programs at the object level.
- a characterisation of structural properties for updates of logic programs.
- two variants of the semantics: *cardinality-preferred* and *set-inclusion preferred* models.
- an equivalence study with proposals from Chapter 6–7.
- a link to a prototype solver that runs online.

This chapter is organised as follows: Section 8.1 is an analysis of the problem that Sakama and Inoue pointed out. The core of the chapter is Section 8.2, which includes a slightly different formulation to the approaches presented in Chapter 7 and shows the main results and comparison with its predecessors in Section 8.3. Next, there is a section reserved to describe an implemented *prototype* of the approach, whose main feature is its characterisation in DLV's *weak constraints*. Finally, concluding remarks are discussed in Section 8.5.

8.1 Problem Description

As introduced in previous chapters, once there existed a strong theoretical *basis for belief revision and updates*, a few authors in the field of logic programming proposed mechanisms for updates with a vast analysis, *taxonomy* and comparison of various known semantics [see Eiter et al., 2000a, 2002, 2005; Sakama and Inoue, 2003; Zhang and Foo, 2005, e.g.]. Some others even suggested new principles, like [Alferes et al., 2005, 1999], with alternative logic-programming frameworks. However, owing to the different foundations each semantics has (even within ASP), still some problems arise to meet a significan number of well-accepted *principles for updates* as suggested in Chapter 4–5.

Although it is unlikely to come off with a semantics that may satisfy all of them [Eiter et al., 2002; Osorio and Cuevas, 2007; Sakama and Inoue, 2003], a more general one that fulfils most of widely-accepted principles is still necessary, which is not an easy task.

For instance, one of the main conclusions of researchers studying updates of logic programs in AGM [Alchourrón et al., 1985] theory —and in other principles around it[1]— is the difficulty to satisfy many of them by means of a *non-monotonic framework* like ASP, owing to the *monotonic nature* of the postulates themselves —[Brewka, 2001; Eiter et al., 2002; Osorio and Cuevas, 2007; Sakama and Inoue, 2003]. Nonetheless, Osorio and Cuevas in [Osorio and Cuevas, 2007] achieved an interpretation of six $(\mathsf{R}\dot{+}1)$–$(\mathsf{R}\dot{+}6)$ of the original eight AGM postulates in terms of the *monotonic* $\mathcal{N}_2$-logic, for general "update" operators. They have chosen $\mathcal{N}_2$-logic because it characterises ASP [Ortiz and Osorio, 2005; Pearce, 1999a]. The authors' result in terms of a general

[1] For a nice analysis and compilation of such principles, see [Eiter et al., 2002].

($\mathsf{PK}*\mathbf{1}$) $\Pi * \{\rho\}$ is a program and ρ a rule.

($\mathsf{PK}*\mathbf{2}$) $\Pi_1 * \Pi_2 \vdash_{\mathcal{N}} \Pi_2$.

($\mathsf{PK}*\mathbf{3}$) $\Pi_1 \cup \Pi_2 \vdash_{\mathcal{N}} \Pi_1 * \Pi_2$.

($\mathsf{PK}*\mathbf{4}$) If $\Pi_1 \cup \Pi_2$ has answer sets then $\Pi_1 \cup \Pi_2 \equiv \Pi_1 * \Pi_2$

($\mathsf{PK}*\mathbf{5}$) $\Pi_2 \equiv_{\mathcal{N}_2} \bot$ implies $(\Pi_1 * \Pi_2) \equiv_{\mathcal{N}_2} \bot$.

($\mathsf{PK}*\mathbf{6}$) If $\Pi_1 \equiv_{\mathcal{N}_2} \Pi_2$ then $\Pi * \Pi_1 \equiv \Pi * \Pi_2$

($\mathsf{PK}*\mathbf{7}$) $(\Pi_1 * \Pi_2) \cup \Pi_3 \vdash_{\mathcal{N}} \Pi_1 * (\Pi_2 \cup \Pi_3)$

($\mathsf{PK}*\mathbf{8}$) If $(\Pi_1 * \Pi_2) \cup \Pi_3$ is consistent, then $\Pi_1 * (\Pi_2 \cup \Pi_3) \vdash_{\mathcal{N}} (\Pi_1 * \Pi_2) \cup \Pi_3$

where $\Pi \vdash_{\mathcal{N}} \alpha$ implies that $\Pi \equiv_{\mathcal{N}_2} \Pi \cup \{\alpha\}$.

Table 8.1: AGM in $\mathcal{N}_2$-logic

semantics, however, seems to be inconclusive [Osorio and Cuevas, 2007], as analysed in Section 4.5. Their interpretation are postulates ($\mathsf{PK}*1$)–($\mathsf{PK}*6$) in Table 8.1, which I have completed with postulates ($\mathsf{PK}*7$)-($\mathsf{PK}*8$). Note that their postulates ($\mathsf{PK}*2$), ($\mathsf{PK}*4$) and ($\mathsf{PK}*6$) in Table 8.1 correspond to KM'-postulates ($\mathsf{R}\circ 1$), ($\mathsf{R}\circ 2$) and ($\mathsf{R}\circ 4$), respectively, from Table 2.5.

On the other hand, the nearest proposal (to the best of my knowledge) that seemed[1] to meet most of the existing principles is due to Sakama and Inoue —refer to Section 4.3— who have introduced and interesting *persistence situation* that others fail to represent well for different reasons, as pointed out by themselves. In particular, the main feature of that interesting situation is that Sakama and Inoue's semantics is capable of maintaining a *knowledge base* (coded into a logic program) at an *object level* throughout its own evolution, which is computed in detail in Observation 4.8 from Section 4.3:

Example 8.1. *Consider the following scenario of two updates to an agent's knowledge base, proposed by Sakama and Inoue*[2] *that shows one of the differences between* update sequences *and* updating at the object level. *This scenario starts with two rules stating that the agent is sleeping when the TV is not on; that is watching it when on; and last, that the TV is on at that instant. Next, new knowledge is incorporated that says it is not possible to have the TV on when there is a power failure; and a new observation of a power failure at that* second instant.

[1] To my knowledge, there is no evidence that their semantics satisfies most of them, although it does overcome most of the problems that other semantics present, as shown in Section 4.3, Chapter 5 and Appendix A.

[2] Originally, this example was proposed by [Alferes et al., 1999], but it has been a little modified in [Sakama and Inoue, 2003] to contrast their differences.

The following pair of logic programs encode the situation, where the agent is updating the initial knowledge base Π_1 *with* Π_2*:*

$$\begin{aligned}
\Pi_1: \quad & sleep \leftarrow \neg tvon \\
& watchtv \leftarrow tvon \\
& tvon \leftarrow \top \\
\Pi_2: \quad & pfailure \leftarrow \top \\
& \bot \leftarrow pfailure, tvon
\end{aligned}$$

Note that in the original example in [Alferes et al., 2005, 1999], they have the rule "$\neg tvon \leftarrow pfailure$*" rather than "*$\bot \leftarrow pfailure, tvon$*". However, they use the same "*$\neg$*" symbol in bodies of rules, having a different meaning with respect to its position at a rule.*[1]

After the first update, most proposals like the ones in Chapter 4 and Chapter 7 (provided the necessary adaptations to update constraints in some of them), coincide with Sakama and Inoue's model: $\{sleep, pfailure\}$*, just as one would expect. However, other approaches capable of dealing with multiple updates like the ones in Section 4.1; Section 4.2; Section 7.2; Section 7.4 diverge after the* second update*, as first pointed out by Sakama and Inoue, by forgetting the latest models and* "reviving" old knowledge *that does not* conflict *with the present situation. In this example the problem arises after the update of* Π_2 *to* Π_1*, with yet the new update program*

$$\Pi_3: \quad \sim pfailure \leftarrow \top$$

having the new unexpected model $\{tvon, watchtv, \sim pfailure\}$*, which does not coincide with common intuition, while Zhang's in Section 4.4 cannot deal with* multiple updates*.*

In other words, updating the resulting knowledge base in Example 8.1 with Π_3 *brings back the previous* "forgotten" models *(*conclusions*) from the first update: the TV switches on "itself"* ***again*** *and the agent is* ***watching*** *it as well! —*$\{tvon, watchtv, \sim pfailure\}$*.*

In this way, Sakama and Inoue's approach modifies the original knowledge base *by producing none or more* programs *after each update, rather than a* model of the complete sequence*. After the* program transformation[2]*, their resulting knowledge base(s) may be interpreted in ASP. For instance, in Example 8.1, their semantics has the unique*

[1] For further details on their "¬" in heads, please refer to Section 4.2

[2] Refer to Example 5.3, which extends Sakama and Inoue's original example into their own notation in order to see it in more detail.

resulting program

$$\begin{aligned}\Pi_4: \quad & sleep \leftarrow \neg tvon \\ & watchtv \leftarrow tvon \\ & pfailure \leftarrow \top \\ & \bot \leftarrow pfailure, tvon\end{aligned}$$

that, if interpreted in ASP, has the ***model***

$$\{\sim pfailure, sleep\} \tag{8.1}$$

that happens to be more intuitive *than its counterparts models, according to Sakama and Inoue, and does meet a* minimal-change principle *as they themselves state, relative to its* non-sequence update *"level".*

Updating at the *object level* is an argument Sakama and Inoue seem to have had against most sequential approaches, that has to do with a *minimal change principle*. However, their *minimal-change principle* is syntactic: the changes to the knowledge base are to be minimal. Moreover, they have no characterisation of their semantics with further *belief-change principles*, arguing that they are not applicable to nonmonotonic propositional theories in general [Sakama and Inoue, 2003].

In addition to that, one big disadvantage of *syntactical approaches* already discussed in previous chapters is that, in general, they do not satisfy the *structural properties* proposed in Section 6.2, and Sakama and Inoue's approach is not an exception —See Chapter 5 and Appendix A.

Regardless the polemic that approach might cause (especially in *planning* domains, like in Example 4.9, Section 4.3) and the deployment of *extended-abduction properties* in their article, the lack of further and more *general properties for belief change* makes it hard to compare with other alternatives for updates of logic programs. Indeed, their first goal (as they themselves explain) is the converse: to provide a mechanism of updates to characterise their *extended abduction framework*, introduced in Section 4.3. Additionally, they characterise different *kinds of updates* (mentioned in Section 4.3) with their framework, claiming that they can provide an *algebra of rule deletion*, besides the addition of them, in order to *explain observations*. Such a syntactic characterisation may have disadvantages in theory of change, because it may leed to information loss, as in Observation 4.9 and Observation 4.10.

On the other hand, approaches like LUPS, briefly described in Section 4.2, might cope well with this situation that explicitly states which rules are persistent and which are not, even if the language approach is sequencial. However, LUPS and DyLP from [Alferes et al., 2005, 1999] in general cannot deal with contradictions that do not depend upon conflicting heads of rules. Take for instance, $\{p \leftarrow q\}$ updated with $\{q \leftarrow \neg p\}$ which has no *(refined) dynamic stable models*. For further details on the latter, refer to

Section 4.2.

The rest of the mentioned proposals perform *updates in sequences* with the already described *counterintuitive behaviour*, or they are undefined for more than one update.

In need to define a *general semantics* and regardless the difference between belief revision and updates due to Katsuno and Mendelzon in Section 2.4, this chapter includes further results from the basic framework in Chapters 6 within the same studied foundation of *Minimal Generalised Answer Sets* (MGAS hereafter from Section 3.6) and an alternative approach to Chapter 7 with a *simpler general formulation* that likewise performs *multiple updates*, but at the *object level* rather than sequences of updates, which overcomes the kind of problems described in Example 8.1. Moreover, the *simpler semantics* meets the *structural properties for updates* already introduced in Section 6.2, as well as the *satisfaction of five of the six most general belief revision postulates*, from Section 2.3. Finally, as an important component of logic programming towards *automatic reasoning* and potential academic purposes, I also provide a running *solver* for this update semantics in an *online prototype* that is a tool and a *testbed* for further experimental results, properties discovery, and (agent) prototypes in a *unified logic-programming framework*.

8.2 $\otimes_o$-operation

One of the main goals of this proposal is to meet most well-accepted *principles* for updates at the *object level* and in Minimal Generalised Answer Sets (MGAS). Similar to the semantics presented in Section 7, the approach consists of setting up the needed models for the desired properties in an *iterated fashion*, rather than a *sequence of updates*.

In particular, there are two main *advantages* of this approach over those presented in Chapter 4 and Chapter 7. First, it comes out of a well-accepted *theoretical basis* from Chapter 2, rather than mending existing proposals. This foundation has proven to overcome counterintuitive results when incorporating redundant or *tautological information* that has been motivation for new model-based approaches —see Chapter 4. Secondly, the framework is capable of maintaining *epistemic states* at an *object level* (differently from sequences of updating programs, like Chapter 7) for multiple updates that meets an *inertia principle* from Sakama and Inoue. These two advantages, amongst others, are encoded into a simple *generalised semantics* described along this chapter.

To recap from Chapter 6, the process of updating a program with another consists in *relaxing* all rules in the original one with a *unique abducible*. In consequence, there is a resulting transformed *relaxed program* representing a new knowledge base that is part of an abductive program. Next, the interpretation consists of the corresponding GAS's of the abductive program from which there is one or more *minimal sets of abducibles* with respect to their inclusion —MGAS's. Finally, the *update model(s)* are expressed in the original alphabet out of the MGAS's.

In this particular operation, the corresponding abducibles take part of one or more new *knowledge bases*, which in turn, with their semantics, represent *epistemic states* rather than just knowledge bases —see Section 2.3 and Section 8.3.4 for further discus-

sion on epistemic states. The semantics is formally expressed with the following set of definitions.

An *α-relaxed rule* is a rule ρ that is *weakened* by a default-negated atom α in its body: $\mathsf{Head}(\rho) \leftarrow \mathsf{Body}(\rho) \cup \{\neg\alpha\}$. In addition, an *$\alpha$-relaxed program* is a set of α-relaxed rules.

Finally, there is a particular case of ELP from Definition 4.13 that contains facts-only, defined as follows.

On the other hand, a *generalised program* of $\mathcal{A}^*$ is a set of rules of form $\{\ell \leftarrow \top \mid \ell \in \mathcal{A}^*\}$, where $\mathcal{A}^*$ is a given set of literals.

Accordingly, *updating a program with another* consists in transforming an ordered pair of programs into a *single abductive program*, as follows.

Definition 8.1 ($\otimes_o$-update Program)**.** *Given an updating pair of extended logic programs, denoted as $\Pi_1 \otimes_o \Pi_2$, over a set of atoms $\mathcal{A}$; and a set of unique abducibles $\mathcal{A}^*$, such that $\mathcal{A} \cap \mathcal{A}^* = \emptyset$; and the α-relaxed program Π' from Π_1, such that $\alpha \in \mathcal{A}^*$; and the abductive program $\Pi_{\mathcal{A}^*} = \langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$. Its* $\otimes_o$-update program *is $\Pi' \cup \Pi_2 \cup \Pi_G$, where Π_G is a generalised program of $\mathcal{M} \cap \mathcal{A}^*$ for some* minimal generalised answer set *$\mathcal{M}$ of $\Pi_{\mathcal{A}^*}$ and $\otimes_o$ is the corresponding update operator.*

It is obvious that Definition 8.1 allows none or more models. In fact, Corollary 8.2 below shows that the update is always consistent provided that Π_1 is also consistent. Moreover, the number of models is determined by the $\otimes_o$-update program.

Corollary 8.1. *Let Π_G be a generalised program out of a minimal generalised answer set $\mathcal{M}$ from $\Pi_{\mathcal{A}^*}$ and $\mathcal{M}_1$ an answer set of Π_G. The following two statements hold:*

a) $\mathcal{M}_1 = \mathcal{M} \cap \mathcal{A}^*$.

b) $\mathcal{M}_1 \subseteq \mathcal{M}$.

Last but not least, the associated *models $\mathcal{S}$ of the new knowledge base* correspond to the answer sets of a $\otimes_o$-update program as follows.

Definition 8.2 ($\otimes_o$-update Answer Set)**.** *Let $\mathbf{\Pi}_{\otimes_\mathbf{o}} = (\Pi_1 \otimes_o \Pi_2)$ be an update pair over a set of atoms $\mathcal{A}$. Then, $\mathcal{S} \subseteq \mathcal{A}$ is a* $\otimes_o$-answer set of $\mathbf{\Pi}_{\otimes_\mathbf{o}}$ *if and only if $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ for some* minimal generalised answer set *$\mathcal{S}'$ of its $\otimes_o$-update program.*

Intuitively, this formulation establishes an order with respect to the *latest update* —which corresponds to postulates $(\mathsf{R} \circ 1)$ and $(\mathsf{U} \circledast 1)$ on their respective Tables 2.5 and 2.6, also recapped below in Section 8.3.4— and with respect to a *minimal change* when choosing the most preferred: MGAS.

In order to illustrate this formulation, consider the following theory, inspired from Alferes et al. and recapped from Example 7.1, describing some knowledge on the sky.

Example 8.2. *[from Example 7.1] Suppose an agent who believes that* when *it is* day *it is not* night *and vice versa, and that there are* stars *when it is* night *and when there are no* clouds*. Finally, that at the current moment it is a fact that there are no* stars*. This simple story may be coded*[1] *into* Π_1 *as follows:*

$$\begin{aligned}\Pi_1 = \{ day &\leftarrow \neg night \\ night &\leftarrow \neg day \\ stars &\leftarrow night, \neg cloudy \\ \sim stars &\leftarrow \top\}\end{aligned}$$

whose unique answer set is $\{day, \sim stars\}$*. Later, the agent acquires new information stating that* stars *and* constls *(*constellations*) are the same thing, as coded in* Π_2*. As soon as the agent updates* Π_1 *with program*

$$\begin{aligned}\Pi_2 = \{ stars &\leftarrow constls \\ constls &\leftarrow stars\}\end{aligned}$$

the expanded alphabet *of the two programs contains only one new* extra atom *with respect to* Π_1*: constls. As the model of* Π_2 *is obviously the empty answer set, constls is considered* synonym *of stars by means of* Π_2*, and thus the update should not change the* original beliefs*. However, the update yields an* extra answer set *in some of the existing update semantics based on the* causal rejection principle *—Section 4.2:*

$$\{stars, constls, night\}$$

which does not coincide with common intuition*.*

The reason is that, although stars can not be true, introducing constls gives another possibility for stars to be true. Thus, the additional answer set is implied.

In general, these supplementary rules in the update are a conservative extension *[Osorio et al., 2001] to* Π_1*: the original language is extended and all answer sets ought to be extensions of the old answer sets. In this specific situation, constls should be true if and only if stars is true.*

In particular, by updating with $\otimes_o$ operator, the example looks as follows.

Example 8.3. *The resulting abductive program of* Π_1 *and* Π_2 *from Example 8.2, is*

[1] Notice that there are other ways to represent the story. The problem is, however, what to do in this particular situation, when the agent runs across this piece of information.

$\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$ *and coincides with the one of* $\oslash$ *and* $\otimes$*, where*

$$\begin{aligned} \Pi' = \{ day &\leftarrow \neg night, \neg\alpha_1 \\ night &\leftarrow \neg day, \neg\alpha_2 \\ stars &\leftarrow night,\ \neg cloudy, \neg\alpha_3 \\ \sim stars &\leftarrow \neg\alpha_4 \} \end{aligned}$$

and $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$*. Next, its unique* MGAS $\{day, \sim stars\}_{\{\sim\alpha_1, \sim\alpha_2, \sim\alpha_3, \sim\alpha_4\}}$ *leaves only one choice to form a* Π_G*. So, the resulting update program is*

$$\begin{aligned} \{ day &\leftarrow \neg night, \neg\alpha_1 \\ night &\leftarrow \neg day, \neg\alpha_2 \\ stars &\leftarrow night, \neg cloudy, \neg\alpha_3 \\ \sim stars &\leftarrow \neg\alpha_4 \\ stars &\leftarrow constls \\ constls &\leftarrow stars \\ \sim\alpha_1 \leftarrow \top; &\quad \sim\alpha_2 \leftarrow \top \\ \sim\alpha_3 \leftarrow \top; &\quad \sim\alpha_4 \leftarrow \top \} \end{aligned}$$

whose ***update answer set*** *is just* $\{day, \sim stars\}$ *as one would expect.*

This example illustrates our main argument proposing that a semantics for updates should depend upon the *logical contents* and not on the particular *syntax to represent knowledge*, generalised by SC and WIS properties.

Finally, this framework is appropriate to overcome, among many other problems, the one introduced in Example 8.1:

Example 8.4. *Suppose an update of* $\Pi_1 \otimes_o \Pi_2$ *with a new update* Π_3*, as suggested in Example 8.1. First, this* iterated update *may also be expressed in a simple form as* $(\Pi_1 \otimes_o \Pi_2) \otimes_o \Pi_3$*. By definition, the abductive program of the first pair is* $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$*, where*

$$\begin{aligned} \Pi' = \{ &sleep \leftarrow \neg tvon, \neg\alpha_1; \qquad watchtv \leftarrow tvon, \neg\alpha_2; \\ &tvon \leftarrow \neg\alpha_3 \} \end{aligned}$$

and $\mathcal{A}^* = \{\alpha_1, \alpha_2, \alpha_3\}$*, whose unique* MGAS $\{pfailure, sleep, \sim tvon\}_{\{\alpha_3\}}$ *and the corresponding unique* generalised program $\Pi_G = \{\alpha_3 \leftarrow \top\}$*. The* resulting knowledge base

of updating Π_1 *with* Π_2 *is then the* $\otimes'_o$*-update program*

$$\begin{aligned}\Pi = \{ sleep &\leftarrow \neg tvon, \neg\alpha_1\\ watchtv &\leftarrow tvon, \neg\alpha_2\\ tvon &\leftarrow \neg\alpha_3\\ \sim tvon &\leftarrow pfailure\\ pfailure &\leftarrow \top\\ \alpha_3 &\leftarrow \top\}\end{aligned}$$

with the corresponding $\otimes_o$***-update answer set*** $\{pfailure, sleep, \sim tvon\}$.

Next, updating the resulting knowledge base Π *with* Π_3 *leads to a new abductive program* $\langle\, \Pi' \cup \Pi_3, \mathcal{A}^*{}_2 \,\rangle$ *where*

$$\begin{aligned}\Pi' = \{ sleep &\leftarrow \neg tvon, \neg\alpha_1, \neg\alpha_{11}\\ watchtv &\leftarrow tvon, \neg\alpha_2, \neg\alpha_{12}\\ tvon &\leftarrow \neg\alpha_3, \neg\alpha_{13}\\ \sim tvon &\leftarrow pfailure, \neg\alpha_{14}\\ pfailure &\leftarrow \neg\alpha_{15}\\ \alpha_3 &\leftarrow \neg\alpha_{16}\}\end{aligned}$$

with its new set of abducibles $\mathcal{A}^*{}_2 = \{\, \alpha_{11}, \alpha_{12}, \alpha_{13}, \alpha_{14}, \alpha_{15}, \alpha_{16}\}$ *and its corresponding unique* MGAS*:* $\{\sim pfailure, sleep, \alpha_3\}_{\{\alpha_{15}\}}$ *as well as the* $\otimes_o$***-update answer set*** *one would expect:* $\{\sim pfailure, sleep\}$.

This section is an introduction to $\otimes_o$-operator by its formal definition and basic examples. The operation consists in updating an initial knowledge base with another by relaxing the former using the same *relaxation method* described in Chapter 6. In addition, the operation may yield a new knowledge base that allows to perform *iterated updates*, and the rest of the chapter describes some properties and an implemented prototype.

8.3 $\otimes_o$-properties

The following properties of this simple formulation are the main result of this current semantics for *iterated updates* of *epistemic states*. In particular, a basic set of *structural properties* was first introduced in Section 6.2, just like the formulation in Chapter 6, besides other properties from the literature. Accordingly, this section is divided into five parts that show a comparison with operators $\oslash$ and $\otimes$ (Section 8.3.1); the referred

structural properties in Section 8.3.2; a case study of consistency-preservation and consistency restoration in Section 8.3.3; Section 8.3.4 generalises this approach in a set of more *general principles*; and finally, other relevant properties from the literature are presented in Section 8.3.5.

8.3.1 Equivalence

To begin with, one *contribution* of this chapter is to show that this approach coincides with operators $\oslash$ and $\otimes$ in the respective Chapter 6 and Chapter 7 for the case of *single updates* and single updates with cardinality preference. The main difference, however, is the limitation to deal with *multiple updates* that was undefined in Chapter 6 ($\oslash$-operator) and later introduced as a *sequence of updates* in Chapter 7 —$\otimes$-operator. The latter, however, has the disadvantage of producing the *counterintuitive results*, for its meta-level approach, as discussed in Example 8.1.

Before starting with equivalence between operators, a recapitulation of their respective definitions is in order: From Chapter 6, the update operator for *pairs of programs* is

Definition 8.3 ($\oslash$-Update Program)**.** *Given an update pair* $\mathbf{\Pi} = (\Pi_1, \Pi_2)$ *of extended logic programs over a set of atoms* $\mathcal{A}$*, an* update program $\mathbf{\Pi}_\oslash = \Pi_1 \oslash \Pi_2$ *corresponds to the abductive program* $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$*, where* $\mathcal{A}^*$ *extends* $\mathcal{A}$ *by new unique abductive atoms and* Π' *is constructed as follows:*

(i) *all constraints in* Π_1*.*

(ii) *for each non-constraint rule* $\rho \in \Pi_1$ *there is a unique abducible* α *(a new atom) and the rule is replaced by* $\mathsf{Head}(\rho) \leftarrow \mathsf{Body}(\rho), \neg\alpha$*.*

where $\oslash$ *represents the the corresponding* update operator*.*

Next, the corresponding update models come out of the MGAS's from the abductive program as follows.

Definition 8.4 ($\oslash$-update Answer Set)**.** *Let* $\mathbf{\Pi} = (\Pi_1, \Pi_2)$ *be an update pair over a set of atoms* $\mathcal{A}$*. Then,* $\mathcal{S} \subseteq \mathcal{A}$ *is an* update answer set *of* $\mathbf{\Pi}$ *if and only if* $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ *for some* minimal generalised answer set $\mathcal{S}'$ *of* $\mathbf{\Pi}$*.*

while the extension to sequences of logic programs from Chapter 7 is given by a relaxed sequence of programs.

The α*-relaxed program* of a sequence of ELP's, $(\Pi_1, \Pi_2, \ldots, \Pi_n)$, over a set of atoms $\mathcal{A}$, is the set $\Pi' = \Pi'_1 \cup \Pi'_2 \cup \cdots \cup \Pi'_{n-1}$, where Π'_i is the α*-relaxed program* of Π_i by a new unique abducible $\alpha \notin \mathcal{A}$ for each rule $\rho \in \Pi_1 \cup \Pi_2 \cup \cdots \cup \Pi_{n-1}$ with $1 \leq i \leq n-1$, and $(\Pi'_1, \Pi'_2, \ldots, \Pi'_{n-1})$ is the corresponding α*-relaxed sequence.*

Accordingly, the $\otimes$*-update sequence operator* for a sequence of programs is defined as follows:

Definition 8.5 (⊗-Update Program)**.** *Given a ⊗-sequence of update extended logic programs* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$*, with* $n \geq 2$*, over a set of atoms* $\mathcal{A}$ *and its corresponding relaxed program* Π'*, its* ⊗-update program *is the abductive program* $\langle \Pi' \cup \Pi_n, \mathcal{A}^* \rangle$ *where* $\mathcal{A}^*$ *is a set of abducibles, such that* $\mathcal{A} \cap \mathcal{A}^* = \emptyset$*, and* ⊗ *is the corresponding update operator.*

The interpretation of the abductive program from a ⊗-update sequence is given by GAS's and the intended models by *minimal sequenced generalised answer sets* as follows.

Definition 8.6 (Abductive Sequence Order; MSGAS)**.** *Given an update sequence of ELP's,* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$*, over a set of atoms* $\mathcal{A}$ *with* n *as an integer; with its corresponding* α*-relaxed sequence* $(\Pi'_1, \Pi'_2, \ldots, \Pi'_{n-1})$*, where* $\alpha \in \mathcal{A}^*$ *and the* α*-relaxed program* Π'*; and the abductive program* $\Pi_{\mathcal{A}^*} = \langle\, \Pi' \cup \Pi_n, \mathcal{A}^* \,\rangle$*:*

- $w(l) = \begin{cases} 1, & l = 1 \\ w(l-1) \cdot |\mathcal{A}^*| + 1 & l > 1 \end{cases}$ *where* l *is a positive integer.*
- $s(\Delta) = \sum_{l=1}^{n-1} (w(l) \cdot |\{\alpha \mid \alpha \in \mathfrak{L}_{\Pi'_l}\}|)$*, where* $\mathcal{M}(\Delta)$ *is a GAS of* $\Pi_{\mathcal{A}^*}$ *and* $\alpha \in \Delta$*.*
- $\mathcal{M}(\Delta_1) \leq_{\mathcal{S}} \mathcal{M}(\Delta_2)$ *if and only if* $s(\Delta_1) \leq s(\Delta_2)$*.*
- $\mathcal{M}(\Delta_1) \equiv_{\mathcal{S}} \mathcal{M}(\Delta_2)$ *if and only if* $s(\Delta_1) = s(\Delta_2)$*.*
- $\mathcal{M}(\Delta)$ *is a* Minimal Sequenced Generalised Answer Set*, MSGAS, of* $\Pi_{\mathcal{A}^*}$ *if and only if* $\mathcal{M}(\Delta)$ *is minimal with respect to* $\leq_{\mathcal{S}}$*.*

Finally, its ⊗-update answer sets should be expressed in the original language of the update sequence out of the MSGAS's:

Definition 8.7 (⊗-update Answer Set)**.** *Given a ⊗-sequence of updating ELP's* $\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$*, with* $n \geq 2$*, over a set of atoms* $\mathcal{A}$*, the set* $\mathcal{S} \subseteq \mathcal{A}$ *is its* ⊗-update answer set *if and only if* $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ *for some* minimal sequenced generalised answer set $\mathcal{S}'$ *of the sequence, and* ⊗ *is the corresponding* update operator.

Besides this recapitulation, it is necessary to make a slight change to Definition 8.1, with the aim to allow a general equivalence, to have an alternative to $\otimes_o$-update operator. The models of $\otimes'_o$-operator shall be in terms of MGAS's, rather than MGAS's, to make them equivalent.

Definition 8.8 ($\otimes'_o$-update Program)**.** *Given an updating pair of extended logic programs, denoted as* $\Pi_1 \otimes'_o \Pi_2$*, over a set of atoms* $\mathcal{A}$*; and a set of abducibles* $\mathcal{A}^*$*, such that* $\mathcal{A} \cap \mathcal{A}^* = \emptyset$*; and the* α*-relaxed program* Π' *from* Π_1*, such that* $\alpha \in \mathcal{A}^*$*; and the*

abductive program $\Pi_{\mathcal{A}^*} = \langle \Pi' \cup \Pi_2, \mathcal{A}^* \rangle$. *Its* $\otimes'_o$-update program *is* $\Pi' \cup \Pi_2 \cup \Pi_G$, *where* Π_G *is a generalised program of* $\mathcal{M} \cap \mathcal{A}^*$ *for some* minimal sequenced generalised answer set $\mathcal{M}$ *of* $\Pi_{\mathcal{A}^*}$ *and* $\otimes'_o$ *is the corresponding update operator.*

Now, the models of such a program should be in terms of the original alphabet of the update pair.

Definition 8.9 ($\otimes'_o$-update Answer Set)**.** *Let* $\mathbf{\Pi}_{\otimes'_o} = (\Pi_1 \otimes'_o \Pi_2)$ *be an update pair over a set of atoms* $\mathcal{A}$. *Then,* $\mathcal{S} \subseteq \mathcal{A}$ *is an* $\otimes'_o$-answer set of $\mathbf{\Pi}_{\otimes'_o}$ *if and only if* $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ *for some* minimal sequenced generalised answer set $\mathcal{S}'$ *of its* $\otimes'_o$*-update program.*

Note that the purpose of the alternative $\otimes'_o$-operator is both to establish equivalence with $\otimes$-operator and to rapidly implement a prototype in DLV, as shown in Section 8.4.

Finally, operators $\otimes_o$, $\otimes'_o$, $\otimes$ and $\oslash$ are equivalent in pairs of programs, depending on their corresponding *ordering function* and other minor details, as formally expressed in the following proposition.

Proposition 8.1. *Suppose an initial extended logic program* Π_1, *without integrity constraints, updated with any* Π_2. *Then,*

$$\Pi_1 \oslash \Pi_2 \equiv \Pi_1 \otimes_o \Pi_2 \text{ and } \Pi_1 \otimes'_o \Pi_2 \equiv \Pi_1 \otimes \Pi_2$$

proof sketch. $\oslash$*-equivalence*: By Definition 8.1, $\Pi_1 \otimes_o \Pi_2$ has the abductive program $\langle \Pi' \cup \Pi_2, \mathcal{A}^* \rangle$, where Π' is the relaxed program of Π_1 and $\mathcal{M}$ an MGAS of the abductive program. On the other hand, by Definition 8.3, $\Pi_1 \oslash \Pi_2$ has the abductive program $\langle \Pi' \cup \Pi_2, \mathcal{A}^* \rangle$ with the same relaxed program Π' whose MGAS correspond to $\mathcal{M}$. By Definition 8.2, the $\otimes_o$-answer sets of $\Pi_1 \otimes_o \Pi_2$ are exactly the same $\oslash$-answer sets of $\Pi_1 \oslash \Pi_2$. Hence, $\Pi_1 \oslash \Pi_2 \equiv \Pi_1 \otimes_o \Pi_2$.

$\otimes$*-equivalence*: By Definition 8.8, $\Pi_1 \otimes'_o \Pi_2$ has the abductive program $\langle \Pi' \cup \Pi_2, \mathcal{A}^* \rangle$, where Π' is the relaxed program of Π_1 and $\mathcal{M}$ an MSGAS of the abductive program. On the other hand, by Definition 8.5, $\Pi_1 \otimes \Pi_2$ has the abductive program $\langle \Pi' \cup \Pi_2, \mathcal{A}^* \rangle$ with the same relaxed program Π' whose MSGAS correspond to $\mathcal{M}$. By Definition 8.9, the $\otimes'_o$-answer sets of $\Pi_1 \otimes'_o \Pi_2$ are exactly the same $\otimes$-answer sets of $\Pi_1 \otimes \Pi_2$. Hence, $\Pi_1 \otimes \Pi_2 \equiv \Pi_1 \otimes'_o \Pi_2$. □

As a consequence, this semantics inherits most of the assets from operators $\oslash$ and $\otimes$: the characterisation of *model-content updates* that preserves the system from being *counterintuitive* when particular situations of updating with *tautological (inert) information* arise, discussed along this thesis, as well as the set of *structural properties* shown below.

Additionally, DyLP semantics presented in Section 4.3 has no *strong negation*, and therefore they use a *syntax-dependent "default" negation in heads* of rules! to have

a similar effect in updates and to solve some problems of *tautological rules* of form $\ell \leftarrow \ell$, in an atypical way —see (4.1) in Section 4.2 for a deeper analysis. Under these circumstances, a careful analysis and comparison with an approximate translation is needed (similar to the one in [Eiter et al., 2002]) in order to consider Alferes et al.'s semantics founded on ASP, which is out of the scope of this current thesis.

Finally, as an important part of logic programming, it is worth mentioning that all examples presented here have been run and tested using existing *solvers* and they are correct and coincide with the discussions in these sections. Firstly, *Eiter et al.'s solver* is available for downloading at `http://www.kr.tuwien.ac.at/staff/giuliana/project.html#Download`, and I have implemented an *online front-end* for it at `http://www2.in.tu-clausthal.de/~guadarrama/updates/upd.html` with a *graphical user interface*. Secondly, I have implemented a solver for update sequences in GAS, as introduced in Section 7.6, which is available online at `http://www.in.tu-clausthal.de/~guadarrama/updates/seqs.html`. Last, I also implemented a solver for this current $\otimes'_o$-updates proposal at `http://www.in.tu-clausthal.de/~guadarrama/updates/o.html` as an online version, naturally.

8.3.2 $\otimes_o$-structural Properties

Some basic properties from the literature are the following, called "*structural properties for updates*, introduced in Sections 6.2 and 6.2. One of the main *contributions* of this chapter is to guarantee that operators $\otimes_o$ and $\otimes'_o$ satisfy that minimal set of properties that is briefly recapitulated below and included in the summary of Appendix A.

Most of them have also been analysed by [Eiter et al., 2002] amongst many other *principles from the literature* and they should contribute as an initial basis of further research. The properties make one of the main *differences with other approaches* and establishes a common frame of reference to compare further proposals.

$\otimes_o$-SP-2, Initialisation [Eiter et al., 2002]: $\emptyset \otimes_o \Pi \equiv \Pi$.

This property states that the update of an initial empty knowledge base yields just the update itself.

$\otimes_o$-SP-3, Inertia: If Π is consistent, $\Pi \otimes_o \emptyset \equiv \Pi$.

A consistent theory is in effect unless new *evidence* states otherwise.

$\otimes_o$-SP-4, Idempotence [Eiter et al., 2002]: $\Pi \otimes_o \Pi \equiv \Pi$.

This property means that the update of program Π with itself has no effect.

$\otimes_o$-SP-6, Non-interference, WNI: [Eiter et al., 2002]: If Π_1 and Π_2 are programs defined over *disjoint alphabets*, and either both of them have answer sets or not, then $\Pi_1 \otimes_o \Pi_2 \equiv \Pi_2 \otimes_o \Pi_1$.

$\otimes_o$-SP-7, Augmented Update [Eiter et al., 2002]: If $\Pi_1 \subseteq \Pi_2$ then $\Pi_1 \otimes_o \Pi_2 \equiv \Pi_2$.

Updating with additional rules makes the previous update obsolete.

$\otimes_o$-SP-8, Strong Consistency, SC: If $\Pi_1 \cup \Pi_2$ is *consistent*, then $\Pi_1 \otimes_o \Pi_2 \equiv \Pi_1 \cup \Pi_2$.
The update coincides with the union when $\Pi_1 \cup \Pi_2$ is consistent.

$\otimes_o$-SP-9, Weak Irrelevance of Syntax, WIS: Let Π, Π_1, and Π_2 be logic programs under the same language. If $T_{\mathcal{N}_2}(\Pi_1) \equiv_{\mathcal{N}_2} T_{\mathcal{N}_2}(\Pi_2)$ then $\Pi \otimes_o \Pi_1 \equiv \Pi \otimes_o \Pi_2$.

The *set of structural properties* has been proposed as fundamental for a proper update semantics along this thesis, and $\otimes_o$-operator satisfies them. Formally,

Theorem 8.1. *Suppose that* Π, Π_1, Π_2 *and* Π_3 *are ELP. Operators* $\otimes_o$ *and* $\otimes'_o$ *satisfy properties* $\otimes_o$*-SP-2–*$\otimes_o$*-SP-9.*

Proof. This is clearly satisfied by Proposition 8.1. □

This result shall be helpful to simplify further properties in upcoming sections.

8.3.3 Dealing with Inconsistencies

Something desirable in belief operations, as suggested in Section 7.3.1, is *dealing with inconsistencies*, not only when new information contradicts previous one, but also with an *originally-inconsistent knowledge base*. This section consists of a study of *consistency-preservation* and *consistency-restoration* as key properties of $\otimes_o$-operator.

In particular, *Weak Consistency* is similar to Lemma 7.1 and guarantees consistency of the abductive program from an update pair. From now on, a *consistent abductive program* from a $\otimes_o$-update pair shall mean the abductive program with generalised answer sets.

Lemma 8.1 (Weak Consistency View)**.** *Suppose* Π_0 *and* Π_1 *are ELP's and an updating pair* $\Pi_0 \otimes_o \Pi_1$ *with its corresponding abductive program* $\Pi_{\mathcal{A}^*} = \langle \Pi' \cup \Pi_1, \mathcal{A}^* \rangle$*. If* Π_1 *is consistent then* $\Pi_{\mathcal{A}^*}$ *is also consistent.*

sketch. Suppose Π_0 and Π_1 are ELP's and that Π_1 is consistent. This means that the abductive program $\langle \Pi' \cup \Pi_1, \mathcal{A}^* \rangle$ is consistent and implies a generalised answer set $\mathcal{M}(\Delta)$ out of the answer sets of $\Pi' \cup \Pi_1 \cup \{\alpha \leftarrow \top \mid \alpha \in \Delta\}$, which is always consistent. Therefore, if Π_1 is consistent, then $\langle \Pi' \cup \Pi_1, \mathcal{A}^* \rangle$ is also consistent. □

Accordingly, the following result holds.

Corollary 8.2 (Consistency Preservation)**.** *Suppose* Π_0 *and* Π_1 *are ELP's. The update* $\Pi_0 \otimes_o \Pi_1$ *is consistent if* Π_1 *is consistent.*

This property is known in the literature as *Consistency Preservation* and by Sakama and Inoue as *inconsistency removal*. Note that the sole name of the latter confirms the *syntactical orientation* of their approach. Finally, notice that this property is equivalent to postulate (R ∘ 3) on Table 2.5.

Finally, the abductive program from $\Pi_0 \otimes_o \Pi_1$ with no abducibles corresponds to the union of the updating pair:

Lemma 8.2. *Suppose* Π_0 *and* Π_1 *are ELP's and an updating pair* $\Pi_0 \otimes_o \Pi_1$ *with its corresponding abductive program* $\langle \Pi'_0 \cup \Pi_1, \mathcal{A}^* \rangle$. *If* $\mathcal{A}^* = \emptyset$ *then* $\Pi'_0 \cup \Pi_1 \equiv_{\mathcal{N}_2} \Pi_0 \cup \Pi_1$.

sketch. Suppose Π_0 and Π_1 are ELP's and that $\mathcal{A}^* = \emptyset$. This means that $\Pi'_0 \cup \Pi_1 \equiv_{\mathcal{N}_2} \Pi_0 \cup \Pi_1$ and thus $\Pi'_0 \cup \Pi_1 \cup \Pi_x \equiv_{\mathsf{ASP}} \Pi_0 \cup \Pi_1 \cup \Pi_x$ for any program Π_x. Additionally, there are two cases for the abductive program $\Pi_{\mathcal{A}^*} = \langle \Pi'_0 \cup \Pi_1, \emptyset \rangle$: That it is consistent or inconsistent. $\Pi_{\mathcal{A}^*}$ consistent means there is a generalised answer set $\mathcal{M}(\emptyset)$ out of the answer sets of $\Pi'_0 \cup \Pi_1 \cup \emptyset$ where Π'_0 coincides with Π_0 and means that both Π_0 and Π_1 must also be consistent, which clearly satisfies the equivalence $\Pi'_0 \cup \Pi_1 \cup \Pi_x \equiv_{\mathsf{ASP}} \Pi_0 \cup \Pi_1 \cup \Pi_x$ for any program Π_x. On the other hand, $\Pi_{\mathcal{A}^*}$ inconsistent means that $\Pi'_0 \cup \Pi_1 \cup \emptyset$ is inconsistent, where Π'_0 coincides with Π_0, which also satisfies the equivalence $\Pi'_0 \cup \Pi_1 \cup \Pi_x \equiv_{\mathsf{ASP}} \Pi_0 \cup \Pi_1 \cup \Pi_x$ for any program Π_x. In either case, $\Pi'_0 \cup \Pi_1 \cup \Pi_x \equiv_{\mathsf{ASP}} \Pi_0 \cup \Pi_1 \cup \Pi_x$ for any program Π_x as required, and therefore $\Pi'_0 \cup \Pi_1 \equiv_{\mathcal{N}_2} \Pi_0 \cup \Pi_1$ if $\mathcal{A}^* = \emptyset$. □

On the other hand, Π_1 inconsistent in Corollary 8.2 may lead to two possible situations: that the resulting update is either consistent or inconsistent, as shown in the following example.

Example 8.5 (Inconsistent Update)**.** *Suppose the update* $\Pi_1 = \{a \leftarrow \neg a\}$*, which has no answer sets, to an original* empty knowledge base $\Pi_0 = \emptyset$*. As a result,* $\bot \models \Pi_0 \otimes_o \Pi_1$*. Now suppose the same update to an initial knowledge base* $\Pi'_0 = \{a \leftarrow \top\}$*. The* $\otimes_o$*-update answer set of such an update* $\{a\} \models \Pi'_0 \otimes_o \Pi_1$.

Corollary 8.2 also proves to be useful when satisfying *belief revision postulates* and to *restore consistency* from an originally *inconsistent knowledge base*. This property is a general case of Sakama and Inoue's *inconsistency removal* that makes syntactical changes to the original knowledge base. Note that their property requires that $\Pi_0 \otimes_o \Pi_1 \subseteq \Pi_0$, contrasting with $\otimes_o$-operator that is a *model-oriented approach*.

As a result, the following proposition follows directly from Corollary 8.2.

Proposition 8.2 (Consistency Restoration)**.** *Suppose* Π_0 *is an ELP. The update* $\Pi_0 \otimes_o \emptyset$ *is consistent.*

As described in this section, $\otimes_o$-operator guarantees *robustness of knowledge bases* in many situations that may break down other alternative frameworks. Accordingly, the properties presented in this section shall be part of a more general framework of principles and postulates.

Notice that Lemma 8.1, together with the resulting corollary and Proposition 8.2 are properties that suggest a classification of $\otimes_o$-operation as a framework for *belief*

revision rather than *belief update*.[1] Notwithstanding, one of the main goals of this work is *representation of knowledge* in general, by defining a framework with an *intuitive behaviour*, even though the framework in this chapter is called "updates" for historical and practical reasons. Having said that, in this thesis no strict distinction shall be made with respect to *belief revision* and *belief update*. It is also worth noticing that Sakama and Inoue have a similar position with respect to such a difference in update semantics, where they suggest ambiguous results may arise when trying to state a clear difference under some contexts [Sakama and Inoue, 2003].

Next, the main core of this chapter is the introduction of a particular interpretation of one of the latest formulations of the AGM-postulates and which of them are met by $\otimes_o$-operator.

8.3.4 $\otimes_o$-principles

One of the main goals of this chapter is a formulation of a semantics for updates of logic programs that can meet as-many-as possible general properties. So, let us start this section with a "particular" interpretation and characterisation of KM′-postulates (R∘1)–(R∘6), presented in Table 2.5 and briefly recapitulated below, as a main foundation to *revising epistemic states*, in this case, encoded into logic programs in $\mathcal{L}_{\mathsf{ASP}}$.

A particular interpretation to a redefinition of AGM-postulates, as fully described in Section 2.3, is due to Darwiche and Pearl, proposing that besides the necessary machinery to reason, an epistemic state should also include the specifications to perform change, as shown with KM′-postulates in Table 2.5.

In a particular context of logic programs, an *epistemic state* is typically represented by a logic program (in this case, an EDLP), Π. Accordingly, $\Pi_0 \circ \Pi_1$ results in a new epistemic state, where each program has an associated finite belief set $Bel(\Pi)$, and a *knowledge base* for each of them $\mathcal{K} \subseteq Bel(\Pi)$, such that $Bel(\Pi)$ is a logical consequence of $\mathcal{K}$ and "$\circ$" is a *generic revision operator* over epistemic states. In particular, the language of the logic program is $\mathcal{L}_{\mathsf{ASP}}$, such that the finite *belief sets* of the program are logically closed in $\mathcal{N}_2$-logic, as in Theorem 3.1. On the other hand, $\Pi_1 \equiv_{\mathsf{ASP}} \Pi_2$ means that both Π_1 and Π_2 have the same answer sets, while $Bel(\Pi_1) \equiv_{\mathcal{N}_2} Bel(\Pi_2)$ means that their corresponding belief sets are equivalent in $\mathcal{N}_2$-logic. Last, an epistemic state is *consistent* if and only if it has *answer sets*. As a result, KM′-postulates may be defined as postulates (RG ∘ 1)–(RG ∘ 6) in Table 8.2.

Postulate (RG ∘ 1) states that *new evidence* should be retained after the update and should have more preference over previous information; while (RG ∘ 2) corresponds to *Strong Consistency* in Section 8.3.2. Next, (RG ∘ 3) prevents from introducing unwarranted inconsistency and is equivalent to consistency preservation in Corollary 8.2; and (RG ∘ 4) corresponds to Dalal's *Principle of Irrelevance of Syntax*, introduced in Section 6.2 and characterising operators $\oslash$, $\otimes$ and $\otimes_o$ to update ASP programs; and an obvious trimmed postulate from (RG ∘ 4) is postulate (RG ∘ 4′). Finally, (RG ∘ 5) and (RG ∘ 6) are specifications of a *minimal change*.

[1] Further details on such differences may be found in Section 2.4.

(**RG** $\circ$ **1**) $\Pi_1 \subseteq \Pi \circ \Pi_1$.

(**RG** $\circ$ **2**) If $\Pi \cup \Pi_1$ is consistent, then $\Pi \circ \Pi_1 \equiv_{\mathsf{ASP}} \Pi \cup \Pi_1$.

(**RG** $\circ$ **3**) If Π_1 is consistent, then $\Pi \circ \Pi_1$ is also consistent.

(**RG** $\circ$ **4**) If $\Pi_x = \Pi_y$ and $\Pi_1 \equiv_{\mathcal{N}_2} \Pi_2$ then $\Pi_x \circ \Pi_1 \equiv_{\mathsf{ASP}} \Pi_y \circ \Pi_2$.

(**RG** $\circ$ **4′**) If $\Pi_1 \equiv_{\mathcal{N}_2} \Pi_2$ then $\Pi \circ \Pi_1 \equiv_{\mathsf{ASP}} \Pi \circ \Pi_2$.

(**RG** $\circ$ **5**) $\Pi \circ (\Pi_1 \cup \Pi_2) \subseteq (\Pi \circ \Pi_1) \cup \Pi_2$.

(**RG** $\circ$ **6**) If $(\Pi \circ \Pi_1) \cup \Pi_2$ is consistent, then $(\Pi \circ \Pi_1) \cup \Pi_2 \subseteq \Pi \circ (\Pi_1 \cup \Pi_2)$.

Table 8.2: Postulates for belief revision of *logic programs*.

An immediate result is the following main theorem of this chapter certifying that $\otimes_o$-operator satisfies most of these belief revision postulates.

Theorem 8.2 (RG $\circ$ -properties)**.** *Suppose that* Π, Π_1 *and* Π_2 *are ELP. Update operator "*$\otimes_o$*" satisfies properties* (*RG* $\circ$ *1*)–(*RG* $\circ$ *4*) *and* (*RG* $\circ$ *6*)*.*

Proof sketch. (**RG** $\circ$ **1**) $\Pi_1 \subseteq \Pi \otimes_o \Pi_1$.

By Definition 8.1, $\Pi_1 \subseteq \Pi' \cup \Pi_G \cup \Pi_1$ that clearly satisfies the objective.

(**RG** $\circ$ **2**) If $\Pi \cup \Pi_1$ is consistent, then $\Pi \otimes_o \Pi_1 \equiv \Pi \cup \Pi_1$.

This postulate corresponds to property $\otimes_o$-SP-8 and satisfied by Theorem 8.1.

(**RG** $\circ$ **3**) If Π_1 is consistent, then $\Pi \otimes_o \Pi_1$ is also consistent.

This postulate is equivalent to Corollary 8.2.

(**RG** $\circ$ **4**) If $\Pi_1 \equiv_{\mathcal{N}_2} \Pi_2$ then $\Pi \otimes_o \Pi_1 \equiv \Pi \otimes_o \Pi_2$.

This postulate is equivalent to property $\otimes_o$-SP-9 in Section 8.3.2 and satisfied by Theorem 8.1.

(**RG** $\circ$ **6**) If $(\Pi \otimes_o \Pi_1) \cup \Pi_2$ is consistent, then $(\Pi \otimes_o \Pi_1) \cup \Pi_2 \subseteq \Pi \otimes_o (\Pi_1 \cup \Pi_2)$.

Suppose $(\Pi \otimes_o \Pi_1) \cup \Pi_2$ is consistent. Then, the abductive program of $\Pi \otimes_o \Pi_1$ is $\langle\, \Pi' \cup \Pi_1, \mathcal{A}^* \,\rangle$ with its respective MGAS's $\mathcal{M}(\Delta_1)$ and its generalised program Π_{G_1}. On the other hand, $\Pi \otimes_o (\Pi_1 \cup \Pi_2)$ has the abductive program $\langle\, \Pi' \cup (\Pi_1 \cup \Pi_2), \mathcal{A}^* \,\rangle$ with its MGAS's $\mathcal{M}(\Delta_2)$ and its generalised program Π_{G_2}. By GAS's definition, $\Delta_1 \subseteq \Delta_2 \subseteq \mathcal{A}^*$ and thus $\Pi_{G_1} \subseteq \Pi_{G_2}$. In consequence, the respective $\otimes_o$-update programs of each pair are $\Pi' \cup \Pi_1 \cup \Pi_2 \cup \Pi_{G_1}$ and $\Pi' \cup \Pi_1 \cup \Pi_2 \cup \Pi_{G_2}$, where

$\Pi' \cup \Pi_1 \cup \Pi_2 \cup \Pi_{G_1} \subseteq \Pi' \cup \Pi_1 \cup \Pi_2 \cup \Pi_{G_2}$ as required. Therefore, $(\Pi \otimes_o \Pi_1) \cup \Pi_2 \subseteq \Pi \otimes_o (\Pi_1 \cup \Pi_2)$.

□

Nevertheless, postulate (RG ∘ 5) does not hold. As a counterexample, consider the following programs: $\Pi = \{a \leftarrow \top; {\sim}b \leftarrow \top; {\sim}c \leftarrow \top\}; \Pi_1 = \{b \leftarrow \top\}; \Pi_2 = \{c \leftarrow \top\}$. Such an update inverts the direction of the relation.

Finally, as $\otimes_o$-operator updates *non-monotonic theories* represented by ASP-programs, resulting in a new non-monotonic theory, one must guarantee strong consistency in postulates (RG ∘ 2) and (RG ∘ 4) of the resulting updated program, which in turn might be updated. Accordingly, the refined postulates are as follows:

(**RG ∘ 2′**) If $\Pi \cup \Pi_1$ is consistent, then $\Pi \circ \Pi_1 \equiv_{\mathcal{N}_2} \Pi \cup \Pi_1$.

(**RG ∘ 4″**) If $\Pi_1 \equiv_{\mathcal{N}_2} \Pi_2$ then $\Pi \circ \Pi_1 \equiv_{\mathcal{N}_2} \Pi \circ \Pi_2$.

Theorem 8.3 (RG ∘ -properties)**.** *Suppose that* Π, Π_1 *and* Π_2 *are ELP. Update operator "*$\otimes_o$*" satisfies properties* (*RG ∘ 2′*) *and* (*RG ∘ 4″*)*.*

Proof sketch. (**RG ∘ 2′**) If $\Pi \cup \Pi_1$ is consistent, then $\Pi \circ \Pi_1 \equiv_{\mathcal{N}_2} \Pi \cup \Pi_1$.

Suppose $\Pi \cup \Pi_1$ consistent. That means $\Pi \circ \Pi_1 \equiv_{\mathcal{N}_2} \Pi \cup \Pi_1$ and $(\Pi \circ \Pi_1) \cup \Pi_x \equiv_{\mathsf{ASP}} \Pi \cup \Pi_1 \cup \Pi_x$ for any program Π_x and $\Pi' \cup \Pi_1 \cup \Pi_x \equiv_{\mathsf{ASP}} \Pi \cup \Pi_1 \cup \Pi_x$ and $\Pi' \cup \Pi_1 \cup \Pi_G \cup \Pi_x \equiv_{\mathsf{ASP}} \Pi \cup \Pi_1 \cup \Pi_x$. As $\Pi \cup \Pi_1$ is consistent, $\mathcal{A}^* \cap \mathfrak{L}_{\Pi'}$ are never positive and $\Pi_G = \emptyset$ and $\Pi' \cup \Pi_1 \cup \Pi_x \equiv_{\mathcal{N}_2} \Pi \cup \Pi_1 \cup \Pi_x$. Then Π' coincides with Π and clearly $\Pi' \cup \Pi_1 \cup \Pi_x \equiv_{\mathsf{ASP}} \Pi \cup \Pi_1 \cup \Pi_x$. Thus $\Pi' \cup \Pi_1 \equiv_{\mathcal{N}_2} \Pi \cup \Pi_1$ and $\Pi \circ \Pi_1 \equiv_{\mathcal{N}_2} \Pi \cup \Pi_1$ as required. Therefore, if $\Pi \cup \Pi_1$ is consistent, $\Pi \circ \Pi_1 \equiv_{\mathcal{N}_2} \Pi \cup \Pi_1$.

(**RG ∘ 4″**) If $\Pi_1 \equiv_{\mathcal{N}_2} \Pi_2$ then $\Pi \circ \Pi_1 \equiv_{\mathcal{N}_2} \Pi \circ \Pi_2$.

Suppose $\Pi_1 \equiv_{\mathcal{N}_2} \Pi_2$. Then $\Pi \circ \Pi_1 \equiv_{\mathcal{N}_2} \Pi \circ \Pi_2$ and $\Pi' \cup \Pi_1 \cup \Pi_G \equiv_{\mathcal{N}_2} \Pi' \cup \Pi_2 \cup \Pi_G$ and $\Pi' \cup \Pi_1 \cup \Pi_G \cup \Pi_x \equiv_{\mathsf{ASP}} \Pi' \cup \Pi_2 \cup \Pi_G \cup \Pi_x$, which is clearly satisfied. Therefore, if $\Pi_1 \equiv_{\mathcal{N}_2} \Pi_2$ then $\Pi \circ \Pi_1 \equiv_{\mathcal{N}_2} \Pi \circ \Pi_2$.

□

8.3.5 Other Properties

There are other relevant and interesting general properties for binary operators that may have a significance in its implementation. As a result, this section includes associativity and distributivity for $\otimes_o$-operator.

Theorem 8.4 (Binary Properties)**.** *Suppose* Π_1, Π_2 *and* Π_3 *are ELP's.* $\otimes_o$*-operator satisfies the following properties:*

Associativity: $\Pi_1 \otimes_o (\Pi_2 \otimes_o \Pi_3) \equiv_{\mathcal{N}_2} (\Pi_1 \otimes_o \Pi_2) \otimes_o \Pi_3$.

Distributivity: $\Pi_1 \otimes_o (\Pi_2 \cup \Pi_3) \equiv_{\mathcal{N}_2} (\Pi_1 \otimes_o \Pi_2) \cup (\Pi_1 \otimes_o \Pi_3)$.

Weak Commutativity: *If* $\Pi_2 \cup \Pi_3$ *is consistent,* $\Pi_1 \otimes_o \Pi_2 \equiv_{\mathcal{N}_2} \Pi_2 \otimes_o \Pi_1$.

Note that Distributivity is a particular case of postulate ($\mathsf{RG} \circ \mathsf{5}$) on Table 8.2.

Proof sketch. Suppose Π_1, Π_2 and Π_3 are ELP's. Then,

Associativity: $\Pi_1 \otimes_o (\Pi_2 \otimes_o \Pi_3) \equiv_{\mathcal{N}_2} (\Pi_1 \otimes_o \Pi_2) \otimes_o \Pi_3$.

The corresponding update programs satisfy the equivalence relation

$$\Pi_1' \cup \Pi_2' \cup \Pi_3 \cup \Pi_{G_a} \cup \Pi_{G_c} \equiv_{\mathcal{N}_2} \Pi_1'' \cup \Pi_2' \cup \Pi_{G_b} \cup \Pi_3$$

By Strong Equivalence,

$$\Pi_1' \cup \Pi_2' \cup \Pi_3 \cup \Pi_{G_a} \cup \Pi_{G_c} \cup \Pi_1'' \cup \Pi_{G_b} \equiv \Pi_1'' \cup \Pi_2' \cup \Pi_{G_b} \cup \Pi_3 \cup \Pi_{G_a} \cup \Pi_{G_c} \cup \Pi_1'$$

that clearly satisfies the objective.

Distributivity: $\Pi_1 \otimes_o (\Pi_2 \cup \Pi_3) \equiv_{\mathcal{N}_2} (\Pi_1 \otimes_o \Pi_2) \cup (\Pi_1 \otimes_o \Pi_3)$. The corresponding update programs satisfy the equivalence relation

$$\Pi_1' \cup \Pi_2 \cup \Pi_3 \cup \Pi_{G_a} \equiv_{\mathcal{N}_2} \Pi_1' \cup \Pi_2 \cup \Pi_{G_b} \cup \Pi_1' \cup \Pi_3 \cup \Pi_{G_c}$$

By Strong Equivalence,

$$\Pi_1' \cup \Pi_2 \cup \Pi_3 \cup \Pi_{G_a} \cup \Pi_{G_b} \cup \Pi_{G_c} \equiv \Pi_1' \cup \Pi_2 \cup \Pi_3 \cup \Pi_{G_a} \cup \Pi_{G_b} \cup \Pi_{G_c}$$

that clearly satisfies the objective.

Weak Commutativity: If $\Pi_2 \cup \Pi_3$ is consistent, $\Pi_1 \otimes_o \Pi_2 \equiv_{\mathcal{N}_2} \Pi_2 \otimes_o \Pi_1$.

Assume $\Pi_2 \cup \Pi_3$ consistent. By postulate ($\mathsf{RG} \circ \mathsf{2}'$),

$$\Pi_1 \otimes_o \Pi_2 \equiv_{\mathcal{N}_2} \Pi_1 \cup \Pi_2 \equiv_{\mathcal{N}_2} \Pi_2 \cup \Pi_1 \equiv_{\mathcal{N}_2} \Pi_2 \otimes_o \Pi_1$$

that clearly satisfies the objective.

Therefore, $\otimes_o$-operator satisfies Associativity, Distributivity and Weak Commutativity. □

This section is an introduction to general properties characterising $\otimes_o$-operator that go from the *structural properties* studied along this thesis, most of them inherited from its equivalent counterparts $\oslash$ and $\otimes$-operators, to more general ones encoded in a particular interpretation of *belief revision postulates*. The satisfaction of AGM-postulates in ASP is something new and important, provided that other current approaches either do not meet most of them or have discarded them for considering that their *monotonic nature* is incompatible with *non-monotonic frameworks* like ASP, as previously discussed in Section 8.1.

The section is also a *study of inconsistencies* not only due to new information that contradicts current knowledge, but also from both an *originally-inconsistent knowledge base*, as well as *new originally-inconsistent observations* that not necessarily contradict current beliefs. The former is something that may be considered a key feature of belief revision. However, as one of the main goals of this thesis is to provide a general framework to correctly represent knowledge, making a strict distinction with belief update theory might result controversial.

On the other hand, *dealing with originally-inconsistent observations* might seem counterintuitive to some researches, but it does not mean that observing such contradictions may not be possible in a *changing environment*. Take for example two *concurrent observations* that contradict each other, updating a current knowledge base. Another example is an *observation that is inconsistent* due to a *typo* or another kind of human error. Traditionally, those problems are left to future debugging, but with a tendency to model even-more *autonomous entities*, tolerating inconsistencies is not only reasonable, but also necessary to *preserve a knowledge base from collapse*.

Finally, the rest of the chapter shows directions to implement a fully-functional prototype that might be helpful to automatically compute updates by means of this approach in the classroom and in a practical laboratory to test properties.

8.4 $\otimes'_o$-prototype

There are some applications where a *credulous semantics* like $\otimes_o$-operator is not desired. This section is an introduction to a more-*skeptical semantics* to $\otimes_o$, called $\otimes'_o$, as well as general directions to implement a rapid prototype in DLV-solver[1]and a short analysis of its complexity. But before going straight to the point, it is necessary a short recapitulation from Section 3.4 and Section 7.6.2 in previous chapters.

8.4.1 Implementing Updates on DLV

As already presented in Section 3.1, there are two major efficient *solvers* to compute ASP with a vast background of implementation and research. They are DLV and SMODELS and the system proposed in this section employs the former at a higher abstraction level in order to *update ELP programs*. Towards this end, this section is an introduction to

[1]This DLV-solver has been presented in [Calimeri et al., 2002; Leone et al., 2006].

a transformation that may be interpreted in either system with some slight syntactic adaptations[1].

To begin with, an approach to implement an update semantics in MGAS has already been introduced in Chapter 6 by means of *preferred disjunctive logic programs* in Brewka's ODLP and has a *solver for pairs of programs* at `http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html`, as described in Section 6.3. However, the mentioned update semantics and thus the final system itself, are limited to *single updates*.

Indeed, a justification to use ODLP is that there is a solver available named `PSmodels`[2] that is an extension to `SMODELS` to compute *preferred answer sets*. Unfortunately, up to the printout of this chapter, there is no reliable version of `PSmodels` and the latest one (v.2.26a) endures some few *bugs* under certain circumstances. In addition to the *running solver*, it is believed that `DLV` significantly outperforms `SMODELS`, not to mention that ODLP is such a colossal semantics that can do much more complex tasks than just computing MGAS's, which might compromise the performance of the desired system and mix up its simple formulation.

As a result this section is an introduction to the use of `DLV`'s *Weak Constraints* for their characteristics of *optimisation in preferences* [Leone et al., 2006] that enjoys the above benefits of being in `DLV` with no more extra throughput added to the system.

Leone et al. introduced a nice feature of `DLV` solver known as *Weak Constraints* that may be employed to set up preferences between models. In particular, a *weak constraint* is a variant of an *integrity constraint* that may be violated in order to establish priorities amongst models. One of its differences is the introduction of a new derivation symbol "$:\sim$", rather than "$:-$ " or "$\leftarrow$". Moreover, one can specify the priority level and weight of the constraint. Formally,

Definition 8.10 (Weak Constraint [Leone et al., 2006])**.** *A* weak constraint *(ω) is an expression of the form*

$$:\sim \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m[w : p] \tag{8.2}$$

where for $0 \leq k \leq m$, $\ell_1, \ldots, \ell_m$ are literals, while w (the weight) and p (the level, or layer) are positive integer constants or variables.

In addition, $\Omega(\Pi)$ shall denote the finite *set of weak constraints* occurring in a given program Π. Likewise, a *ω-program* is a logic program with weak constraints.

In order to provide a more syntactic sugar, another way to define a weak-consitraint expression from Definition 8.10 is as follows.

Definition 8.11 (Weak Constraint)**.** *A* weak constraint *(ω) is an expression of the form*

$$[w : p] \leftarrow \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m \tag{8.3}$$

[1] Refer, for instance, to [Leone et al., 2006] for an *equivalence of weak constraints* in `SMODELS`.

[2] `http://www.tcs.hut.fi/Software/smodels/priority/`

where for $0 \leq k \leq m$, $\ell_1, \ldots, \ell_m$ *are literals, while* w *(the weight) and* p *(the level, or layer) are positive integer constants or variables.*

From now on, the previous weak-constraints form shall be employed in the context of DLV-code, while the other in higher abstraction levels.

Similarly to integrity constraints in Section 3.1, one may say that a weak constraint $\rho = ([w : p] \leftarrow \ell_1, \ldots, \ell_k, \neg\ell_{k+1}, \ldots, \neg\ell_m)$ is *violated* by an answer set $\mathcal{S}$ of a program Π if the following three conditions hold:

1. $\rho \in \Pi$
2. $\{\ell_1, \ldots, \ell_k\} \subseteq \mathcal{S}$
3. $\{\ell_{k+1}, \ldots, \ell_m\} \nsubseteq \mathcal{S}$

Additionally, Leone et al. simplify the combination of weights in levels by introducing a function $H^{\Pi}(\mathcal{S})$ that grows in direct proportion to the weight and level of the weak constraint as follows:

Definition 8.12 (Objective Function, $H^{\Pi}(\mathcal{S})$ [Leone et al., 2006])**.** *Given a ground program* Π *with weak constraints* $\Omega(\Pi)$ *and an answer set* $\mathcal{S}$*, the* ω objective function $H^{\Pi}(\mathcal{S})$ *is defined by using an auxiliary function* f_{Π} *that maps levelled weights to weights without levels:*

$$\begin{aligned} f_{\Pi}(1) &= 1 \\ f_{\Pi}(n) &= f_{\Pi}(n-1) \cdot |\Omega(\Pi)| \cdot w^{\Pi}_{max} + 1, n > 1 \\ H^{\Pi}(\mathcal{S}) &= \sum_{i=1}^{l^{\Pi}_{max}} (f_{\Pi}(i) \cdot \sum_{\rho \in N^{\Pi}_i(\mathcal{S})} weight(\rho)) \end{aligned}$$

where $N^{\Pi}_i(\mathcal{S})$ *denotes the weak constraints at level* i *violated by* $\mathcal{S}$*, and* $weight(\rho)$ *the weight of weak constraint* ρ*.*

Finally, the *best models* of such a logic program are those that *minimise* the number of *violated weak constraints.*

Definition 8.13 (Weak-Constraint Model [Leone et al., 2006])**.** *For an EDLP* Π *with weak constraints, a set* $\mathcal{S}$ *is a* weak-constraint model *of* Π *if and only if*

1. $\mathcal{S}$ *is an answer set of* Π
2. $H^{\Pi}(\mathcal{S})$ *is minimal over all the answer sets of* Π*.*

When the underlying semantics is ASP in Definition 8.13, a weak-constraint model is also known as *Optimal Answer Set.*

Moreover, the language of EDLP's with weak constraints shall be called $\mathsf{DATALOG}^{\vee,\omega}$, which is very similar to the notation from the literature.

8.4.2 Weak-constraints Characterisation

This section is an alternative to the characterisation in Section 6.3, and it consists in transforming updates of ELP's into a ω-program. The answer sets of such a program shall prove to coincide with the GAS's of the abductive program from the update. Finally, the *optimal answer sets* of the ω-program shall coincide with the MSGAS's of the abductive program. So, let us start by introducing some more notation.

A *Simple Weak Constraint* (ω') is an expression of the form

$$[w : p] \leftarrow \ell$$

where ℓ is a literal; and w and p are optional weight and level parameters as in Definition 8.10.

A ω' derives the following result with some new notation in advance: read $\mathfrak{L}_{\Omega(\Pi)}$ as the *signature of the weak constraints* occurring in Π (a finite set of literals), while $\mathfrak{L}_{N_1^{\Pi}(\mathcal{M})}$ stands for the *signature* of the weak constraints in Π at level 1 violated by $\mathcal{M}$. Something worth recalling is that $N_i^{\Pi}(\mathcal{M})$ denotes the weak constraints at level i violated by $\mathcal{M}$ in Π.

Proposition 8.3. *Suppose a logic program* Π *with weak constraints* $\Omega(\Pi)$ *of the form of a* simple weak constraint *with* $w = p = 1$*; suppose an answer set* $\mathcal{M}$ *of* Π*; and a set of literals* $\Delta \subseteq \mathcal{M} \cap \mathfrak{L}_{\Omega(\Pi)}$*. Then,* $\mathfrak{L}_{N_1^{\Pi}(\mathcal{M})} = \Delta$*.*

It is worth mentioning that the *model(s) of a logic program with weak constraints* are also called answer sets in the literature.

Before introducing the translation function, let us recap the well-known standard *signature* definition with $\mathfrak{L}_{\langle\, \Pi, \mathcal{A}^* \,\rangle}$, which means the finite set of literals occurring both in Π and in $\mathcal{A}^*$.

Definition 8.14 (Abductive-$\mathcal{W}$ Translation)**.** *Let* $\langle\, \Pi, \mathcal{A}^* \,\rangle$ *be an abductive logic program. A* translation into a weak-constraint program $\mathcal{W}(\Pi, \mathcal{A}^*)$ *corresponds to*

$$\Pi \cup \{\alpha' \vee \alpha \leftarrow \top\} \cup \{[1:1] \leftarrow \alpha\} \tag{8.4}$$

where $\alpha \in \mathcal{A}^*$ *and* $\alpha' \notin \mathfrak{L}_{\langle\, \Pi, \mathcal{A}^* \,\rangle}$*.*

This translation and Proposition 8.3 yield the following useful results for the *implementation* of GAS-semantics in *weak constraints*.

Lemma 8.3. $\mathcal{M} \cap \mathfrak{L}_{\langle\, \Pi, \mathcal{A}^* \,\rangle}$ *is a* generalised answer set of an abductive program $\langle\, \Pi, \mathcal{A}^* \,\rangle$ *if and only if* $\mathcal{M}$ *is an answer set of* $\mathcal{W}(\Pi, \mathcal{A}^*)$*.*

sketch. Only-if part. Suppose $\mathcal{M}$ is an answer set of $\mathcal{W}(\Pi, \mathcal{A}^*)$. By Definition 8.14, there are two cases for the answer sets of the ω-program. That $\mathcal{M}$ is an answer set of

$\Pi \cup \{\alpha' \vee \alpha \leftarrow \top\} \cup \{[1:1] \leftarrow \alpha\}$ with $\alpha \in \mathcal{M}$ and by Definition 3.17, $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ corresponds to the answer set of $\Pi \cup \{\alpha \leftarrow \top\}$ and therefore by Definition 3.17, to the GAS of $\langle \Pi, \mathcal{A}^* \rangle$. On the other hand, with $\alpha \notin \mathcal{M}$, $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is just an answer sets of $\Pi \cup \{\}$ and thus the GAS of $\langle \Pi, \mathcal{A}^* \rangle$ —Definition 3.17. In both cases, $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is a GAS of $\langle \Pi, \mathcal{A}^* \rangle$, as required.

If part. Suppose $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is a GAS of $\langle \Pi, \mathcal{A}^* \rangle$. By Definition 3.17, there are two cases for Δ. That $\Delta \neq \{\}$ means that $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is an answer set of $\Pi \cup \{\alpha \leftarrow \top\}$ for some $\alpha \in \Delta$ with $\Delta \subseteq \mathcal{A}^*$, which corresponds to the answer set $\mathcal{M}$ of the translated program (Definition 8.14) $\Pi \cup \{\alpha' \vee \alpha \leftarrow \top\} \cup \{[1:1] \leftarrow \alpha\}$ with $\alpha \in \mathcal{M}$. On the other hand, $\Delta = \{\}$ means that $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is an answer set of $\Pi \cup \{\}$ that corresponds to the answer set $\mathcal{M}$ of the ω-program $\Pi \cup \{\alpha' \vee \alpha \leftarrow \top\} \cup \{[1:1] \leftarrow \alpha\}$ with $\alpha \notin \mathcal{M}$. In both cases, $\mathcal{M}$ is an answer set of $\mathcal{W}(\Pi, \mathcal{A}^*)$, as required. □

The *equivalence between a GAS of an abductive program and an answer set of a ω-program* ought to be easier to read after a simple example:

Example 8.6. *Suppose an update of* Π_1 *with* Π_2 *where* $\Pi_1 = \{b \leftarrow \top\}; \Pi_2 = \{a \leftarrow \top\}$. *Its corresponding abductive program is* $\langle \Pi' \cup \Pi_2, \mathcal{A}^* \rangle$ *where* $\Pi' = \{b \leftarrow \neg\alpha\}$ *and* $\mathcal{A}^* = \{\alpha\}$. *As a result,* $\mathcal{W}(\Pi' \cup \Pi_2, \mathcal{A}^*) = \Pi' \cup \Pi_2 \cup \{\alpha' \vee \alpha \leftarrow \top\} \cup \{[1:1] \leftarrow \alpha\}$ *whose answer set* $\mathcal{M} = \{\alpha', a, b\}$. *On the other hand, the GAS of the abductive program is just* $\{a, b\}_\emptyset = \mathcal{M} \cap \mathfrak{L}_{\langle \Pi' \cup \Pi_2, \mathcal{A}^* \rangle}$.

Up to now, one can compute GAS's by means of ω-programs. However, this proposal of *updates at the object level* requires that the generalised answer sets are minimal with respect to the number of abducibles, as in Definition 8.6. Thus, the **main result** of this section is the following theorem that shows the *equivalence between MSGAS's and optimal answer sets*, the **core of the implementation** and one more property for the semantics.

Theorem 8.5 (MSGAS–Optimal Answer Set)**.** *Let $\mathcal{M}$ be a set of literals. The set $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is a* minimal sequenced generalised answer set *of the abductive program $\langle \Pi, \mathcal{A}^* \rangle$ if and only if $\mathcal{M}$ is an* optimal answer set *of $\mathcal{W}(\Pi, \mathcal{A}^*)$.*

Proof sketch. If part. Suppose $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is a minimal sequenced generalised answer set of the abductive program $\langle \Pi, \mathcal{A}^* \rangle$. Then, $\mathcal{M} \cap \mathfrak{L}_{\langle \Pi, \mathcal{A}^* \rangle}$ is a GAS of $\langle \Pi, \mathcal{A}^* \rangle$ and is minimal with respect to $\leq_{\mathcal{S}}$ and by Lemma 8.3 $\mathcal{M}$ corresponds to the answer set of $\mathcal{W}(\Pi, \mathcal{A}^*)$ and therefore, to its weak-constraint model —Definition 8.13— that is optimal with respect to cardinality.

Only-if part. Suppose $\mathcal{M}$ is an optimal answer set of $\mathcal{W}(\Pi, \mathcal{A}^*)$. Then, by Definition 8.13, $\mathcal{M}$ is minimal with respect to $H^\Pi(\mathcal{M})$ and corresponds to the minimal set

of violated weak constraints that are exactly the minimal number of abducibles in the generalised answer set $\mathcal{M} \cap \mathfrak{L}_{\langle\, \Pi, \mathcal{A}^* \,\rangle}$ of $\langle\, \Pi, \mathcal{A}^* \,\rangle$ by Proposition 8.3. □

Finally, the update-answer sets of a resulting $\otimes'_o$-update program are the MSGAS's expressed in the original alphabet. Formally,

Definition 8.15 ($\otimes'_o$-update Answer Set)**.** *Let* $\mathbf{\Pi}_{\otimes_o} = (\Pi_1 \otimes'_o \Pi_2)$ *be an update pair over a set of atoms* $\mathcal{A}$*. Then,* $\mathcal{S} \subseteq \mathcal{A}$ *is an* $\otimes'_o$-answer set of $\mathbf{\Pi}_{\otimes_o}$ *if and only if* $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ *for some* minimal sequenced generalised answer set $\mathcal{S}'$ *of its* $\otimes'_o$*-update program.*

Since $\otimes'_o$-answer sets are *minimal sequenced generalised answer sets*, it is a semantics that may be considered more skeptical than the original $\otimes_o$, as illustrated in the following example.

Example 8.7. *Let*

$$\begin{aligned} \Pi_1 &= \{(a \leftarrow \top); (b \leftarrow \top); (c \leftarrow \top)\} \\ \Pi_2 &= \{(\sim a \leftarrow c); (\sim b \leftarrow a); (\bot \leftarrow a, b)\} \end{aligned}$$

The abductive program $\langle\, \Pi'_1 \cup \Pi_2, \mathcal{A}^* \,\rangle$ *has the GAS's*

$$\{a, \sim b, \alpha_2, \alpha_3\}; \{\sim a, b, c, \alpha_1\};$$
$$\{\sim a, c, \alpha_1, \alpha_2\}; \{b, \alpha_1, \alpha_3\};$$
$$\{\alpha_1, \alpha_2, \alpha_3\}$$

and the abductive program has the ***unique MSGAS:*** $\{\sim a, b, c, \alpha_1\}$ *with its obvious corresponding* $\otimes'_o$*-update answer set. On the other hand, the abductive program has* ***two MGAS's.*** *Namely,* $\{\sim a, b, c, \alpha_1\}$ *and* $\{a, b, \alpha_2, \alpha_3\}$*, and its two obvious corresponding* $\otimes_o$*-update answer sets:* $\{\sim a, b, c\}$ *and* $\{a, b\}$*.*

Before analysing the complexity of $\otimes'_o$-operator, let us fully illustrate *more than one update* to an original knowledge base, by means of the following simple example.

Example 8.8. *Let* $\Pi_1 = \{(a \leftarrow \top); (b \leftarrow \top)\}$ *;* $\Pi_2 = \{\sim b \leftarrow \top\}$ *and* $\Pi_3 = \{\sim a \leftarrow \top\}$ *with the updates* $(\Pi_1 \otimes'_o \Pi_2) \otimes'_o \Pi_3$*. The corresponding abductive update program of the first pair is* $\langle\, \Pi'_1 \cup \Pi_2, \mathcal{A}^* \,\rangle$ *where* $\Pi'_1 = \{(a \leftarrow \neg\alpha_1); (b \leftarrow \neg\alpha_2)\}$ *and* $\mathcal{A}^* = \{\alpha_1, \alpha_2\}$*. Next,* $\mathcal{W}(\Pi'_1 \cup \Pi_2, \mathcal{A}^*) = \Pi'_1 \cup \Pi_2 \cup \{\alpha'_1 \vee \alpha_1 \leftarrow \top\} \cup \{[1:1] \leftarrow \alpha_1\} \cup \{\alpha'_2 \vee \alpha_2 \leftarrow \top\} \cup \{[1:1] \leftarrow \alpha_2\}$ *whose answer sets correspond to the GAS's* $\{\sim b, a\}_{\{\alpha_2\}}$ *and* $\{\sim b\}_{\{\alpha_1, \alpha_2\}}$*. Then, the unique MSGAS is just the optimal answer set of the* ω*-program:* $\{a, \sim b, \alpha_2\}$*, whose generalised program* $\Pi_G = \{\alpha_2 \leftarrow \top\}$ *and the new* $\otimes'_o$*-update program is* $\Pi'_1 \cup \Pi_2 \cup \Pi_G$

that corresponds to

$$\Pi = \{(a \leftarrow \neg\alpha_1); (b \leftarrow \neg\alpha_2); (\sim b \leftarrow \top); (\alpha_2 \leftarrow \top)\}$$

The second update is then represented as $\Pi \otimes'_o \Pi_3$ *that has the abductive program* $\langle\, \Pi' \cup \Pi_3, \mathcal{A}^*{}_2 \,\rangle$ *where*

$$\Pi' = \{(a \leftarrow \neg\alpha_1, \neg\alpha_{11}); (b \leftarrow \neg\alpha_2, \neg\alpha_{12}); (\sim b \leftarrow \neg\alpha_{13}); (\alpha_2 \leftarrow \neg\alpha_{14})\}$$

and $\mathcal{A}^*{}_2 = \{\alpha_{11}, \alpha_{12}, \alpha_{13}, \alpha_{14}\}$. *Next, the new* ω*-program*

$$\begin{aligned}
\mathcal{W}(\Pi' \cup \Pi_3, \mathcal{A}^*{}_2) = \{a &\leftarrow \neg\alpha_1, \neg\alpha_{11} \\
\sim\alpha_{11} \vee \alpha_{11} &\leftarrow \top \qquad [1:1] \leftarrow \alpha_{11} \\
b &\leftarrow \neg\alpha_2, \neg\alpha_{12} \\
\sim\alpha_{12} \vee \alpha_{12} &\leftarrow \top \qquad [1:1] \leftarrow \alpha_{12} \\
\sim b &\leftarrow \neg\alpha_{13} \\
\sim\alpha_{13} \vee \alpha_{13} &\leftarrow \top \qquad [1:1] \leftarrow \alpha_{13} \\
\alpha_2 &\leftarrow \neg\alpha_{14} \\
\sim\alpha_{14} \vee \alpha_{14} &\leftarrow \top \qquad [1:1] \leftarrow \alpha_{14} \\
\sim a &\leftarrow \top\}
\end{aligned}$$

whose answer sets correspond to the GAS*'s*

$$\begin{aligned}
\{\sim a, \sim b\}_{\{\alpha_{11},\alpha_2\}}; \quad & \{\sim a, \sim b\}_{\{\alpha_{11},\alpha_2,\alpha_{12}\}}; \\
\{\sim a\}_{\{\alpha_{11},\alpha_2,\alpha_{13}\}}; \quad & \{\sim a\}_{\{\alpha_{11},\alpha_2,\alpha_{12},\alpha_{13}\}}; \\
\{\sim a, \sim b\}_{\{\alpha_{11},\alpha_{12},\alpha_{14}\}}; \quad & \{\sim a, b\}_{\{\alpha_{11},\alpha_{13},\alpha_{14}\}}; \\
\{\sim a\}_{\{\alpha_{11},\alpha_{12},\alpha_{13},\alpha_{14}\}} &
\end{aligned}$$

from which, the unique MSGAS *is just the optimal answer set of the* ω*-program is* $\{\sim a, \sim b, \alpha_2, \alpha_{11}\}$ *and its* $\otimes'_o$*-update answer set is simply* $\{\sim a, \sim b\}$*, as one would expect.*

8.4.3 $\otimes'_o$-complexity

This section is a brief introduction to the *complexity of* $\otimes'_o$ *operator* that, broadly speaking, has its bottleneck in the process of DATALOG$^{\vee,\omega}$. That is to say, the problem of $\otimes'_o$-updating Π_0 with Π_1 may be divided into two basic processes, very similar to

updating sequences in Section 7.6.5. Firstly, there is an implicit simple algorithm in the definition of α-relaxed program that transforms an ELP into a ω-program with disjunctions. Secondly, the resulting propositional (ground) *ω-program* is an ordinary DATALOG$^{\vee,\omega}$ to be computed by DLV, which calculates its ω-models. By following the notation introduced in Section 3.7, the time to compute such a process is bounded by Θ_3^{P} in the size of the ω-program in general [Buccafurri et al., 2000; Leone et al., 2006], while a propositional ELP without weak constraints is known to be Δ_2^{P} in general, assuming that $\mathrm{P} = \text{co-P}; \Delta_2^{\mathrm{P}} = \text{co-}\Delta_2^{\mathrm{P}}; \Delta_3^{\mathrm{P}} = \text{co-}\Delta_3^{\mathrm{P}}; \Theta_3^{\mathrm{P}} \subseteq \Delta_3^{\mathrm{P}}$.

The complexity of computing a propositional DATALOG$^{\vee,\omega}$ program is still polynomial in the size of the input, for the case of a single *ω-component* —single ω-level $N_1^{\Pi}(\cdot)$, in this thesis notation— [Buccafurri et al., 2000; Leone et al., 2006]. For the case of *non-ground input-programs*, however, the complexity of DATALOG$^{\vee,\omega}$ rises to *exponential*, bounded by co-NEXPTIME$^{\mathrm{NP}}$ [Leone et al., 2006]. That is to say, *grounding a program* may produce a ground-program that is exponentially larger than the size of the original non-ground program [Heymans, 2006]. Although the proposals of this thesis are in the *propositional case*, there is no restriction to compute non-ground programs, besides the well-known problem of grounding them: notice that $\otimes_o$-operator does not change the semantics of the initial theory, while $\otimes'_o$-operator may be more *credulous*.

In particular, the *program transformation*, may be reduced to the problem of *tagging* each rule of a program Π_0 with a subset of the natural numbers $(1, 2, \ldots, r)$ by the definition of α-relaxed program, where $r = |\{\rho \mid \rho \in \Pi_0\}|$. As a consequence, the following obvious result holds, like in Proposition 7.7:

Proposition 8.4. *The complexity of transforming any extended logic program Π_0 into an α-relaxed program is linear in the number of rules $r = |\{\rho \mid \rho \in \Pi_0\}|$.*

Notwithstanding, the process of weakening an original knowledge base may tend to be more *expensive* while iterations occur, for such a process will introduce new facts that are true literals in $\otimes'_o$-models. That is to say, a particular application might have databases large-enough over iterations so as to fall down into *program complexity* —*polynomial*— rather than into *data complexity* —*exponential*— or *combined complexity*. Note than in general, *combined complexity* is taken as the main measure, according to Dantsin et al..

Once the original knowledge base has been weakened, there is an ordinary ground ω-program to be interpreted in DATALOG$^{\vee,\omega}$, whose possible interpretations of the weakened rules and introduced abducibles grow *exponentially in the number of rules* of Π_0 by having an upper bound of $2^{|\mathcal{A}^*|}$ possible *combinations to include abducibles* into every interpretation, depending on the algorithm employed to find GAS's —see Example 7.4. In the case of $\otimes'_o$-operator, however, such an algorithm is up to DLV. Finally, as the transformed updating pair is a propositional case (ground) for a *single component*[1] (single ω-level, in our notation), the particular complexity of the whole problem of computing it in DLV is bounded by Θ_3^{P}, which isn't a bad result, according

[1] Buccafurri et al.

to the hierarchy introduced in Section 3.7.1: $\Sigma_2^{\mathrm{P}} \subseteq \Theta_3^{\mathrm{P}} \subseteq \Delta_3^{\mathrm{P}}$. In other words, $\mathrm{NP}^{\mathrm{NP}} \subseteq \Theta_3^{\mathrm{P}} \subseteq \mathrm{P}^{\Sigma_2^{\mathrm{P}}}$ or

$$\mathrm{NP}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{NP}^{\mathrm{NP}[O(\log n)]}} \subseteq \mathrm{P}^{\mathrm{NP}^{\mathrm{NP}}},$$

where n is the input size.

However, given the exponential input size of a grounded relaxed initial knowledge base, the overall complexity lies in the exponential hierarchy with respect to the original non-relaxed pair:

Proposition 8.5. *The complexity of computing a $\otimes'_o$-answer set of any pair of grounded ELP's (Π_0, Π_1) is* exponential *in the number of rules $r = |\{\rho \mid \rho \in \Pi_0\}|$, bounded by co-NEXPTIME^{NP}.*

On the other hand, computing *ungrounded programs* in DATALOG$^{\vee,\omega}$ is very expensive and thus a clear shortcoming for semantics of updates of logic programs in ASP so as to have typical *industrial applications* of, say, *updating large knowledge bases*. Notwithstanding, the implementation to process *toy examples* in the classroom is an immediate practical application, as well as others like a *property testbed* and more complex prototypes, to mention a few, that shouldn't be despised.

In summary, the only complexity result relevant to this thesis is Proposition 8.5, as the complexity of the entire process to find $\otimes'_o$-models depends on the particular application of updating knowledge bases. Notwithstanding, the entire process is not affected by the $\otimes'_o$-operation and is still bounded by the complexity of the corresponding DATALOG$^{\vee,\omega}$: whether it is propositional or ungrounded; *sequencial* or *iterative*; and whether it is data-complete or *combined* with program-completeness.

Having presented the complexity and equivalence between weak-constraint models and MSGAS, the following section consists of a general view of its implementation, which confirms the claims along this thesis and provides a testbed with a link to an *online prototype* for further research on property tests, toy examples and a component of more complex (agent) applications.

8.4.4 The Parser

Differently from the implementation at `http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html`, which has a parser embedded in its PHP[1] code, the new *parser* presented in this section has been compiled in C-language generated by *Lex* and *Yacc* on UNIX and it may be run online, with no need to download it, at: `http://www.in.tu-clausthal.de/~guadarrama/updates/o.html`. The advantage of having a *UNIX binary module* is the ease to be a plug-in to other modules so as to form more complex scenarios.

[1] This is a well-known script language, quite suitable for small processes of dynamic contents for web pages, as well as for *regular-expressions modules*.

The parsing process consists in entering two sets of ELP's (each within braces) that is analysed, decomposed and transformed into a ω-program, whose interpretation is executed by `DLV`.

The BNF grammar and tokens are very similar to the ones in the sequence semantics presented in Section 7.6.3. Consequently, they are omitted in this section.

As mentioned before, once a rule is analysed and weakened, the system constructs a pair of new rules with weakening atoms under *weak-constraints semantics* like Definition 8.14:

$$[1:1] \leftarrow \alpha_i \tag{8.5}$$
$$\sim\alpha_i \mid \alpha_i \leftarrow \top \tag{8.6}$$

The intuition behind this formulation is computing the MGAS of the abductive program by *violating the least number of weak constraints* —refer to Section 3.4.

Example 8.9. *Suppose the update of* Π_1 *by* Π_2 *where* $\Pi_1 = \{a \leftarrow \neg b\}; \Pi_2 = \{(b \leftarrow \neg a); (c \leftarrow a)\}$ *The corresponding abductive program is* $\langle\, \Pi' \cup \Pi_2, \mathcal{A}^* \,\rangle$ *where* $\Pi' = \{a \leftarrow \neg b, \neg\alpha_1\}$ *and* $\mathcal{A}^* = \{\alpha_1\}$. *Such an abductive program is transformed into a preference program as*

```
a:- not b , not alpha_1 .
~alpha_1 | alpha_1 .
:~ alpha_1 . [1:1]
b:- not a .
c:- a .
```

that has two optimal answer sets *from which one may form two identical* update program*s:*

```
{a:- not b , not alpha_1 .
b:- not a .
c:- a .
~alpha_1 .}
```

that have two optimal answer sets:

```
{~alpha_1 , a , c}, {~alpha_1 , b}
```

At this final stage, the resulting programs are *preferred transformed program*s to be interpreted in `DLV` with weak constraints. Such a process yields zero or more *optimal answer sets* to be transformed into MSGAS's by a control script.

8.4.5 Discussion

This section is an introduction of general methods for *rapid prototyping* in DLV of logic programming semantics, as well as directions to implement the declarative version of both an update semantics and MSGAS's. The system has been developed with a strong emphasis on *declarative programming*, in just some few fragments of procedural modules, in order to make it easily modifiable for particular frameworks and to confirm claims of the original semantics here presented. Another of its highlights is its modularity and *UNIX paradigm* that allows it to be a *web service* and easily plugged into other systems via on-line even without needing to download it. Moreover, its simple standard *graphical user interface* in HTML makes it very easy to use, compared to most of the solvers implemented for command-line use.

As one of the main components of Logic Programming, implementation of semantics helps quickly understand it (for educational proposes and for a reliable comparison tool, for instance), spread it, test properties and compute knowledge bases for more complex prototypes and other frameworks. In addition, an analysis on the *complexity* of the prototype shows that the transformation from a given sequence of logic programs is polynomial, while computing the resulting transformed program may be exponential, due to DLV's weak constraints. Nevertheless, the implementation shouldn't be despised, as it has immediate practical academic applications at least.

8.5 Conclusions of Chapter 8

This chapter is an introduction to a new definition for updates at the object level that is more general than the ones presented in previous approaches, overcoming the problem that Sakama and Inoue have pointed out in order to meet a *persistence principle* and a particular *minimality of change* —at the object level— essential for a general semantics for updates. As a result, it is equivalent to the restricted cases of its predecessors, and meets the same basic structural properties.

The core of this chapter shows that the new semantics satisfies the introduced set of *structural properties for sequences* of updating logic programs; a general set of *postulates for belief revision*; as well as other needed properties as *consistency preservation* and *inconsistent updates*. The latter set of properties have to do with *consistency restoration* both from an original inconsistent knowledge base and an inconsistent observation that is in turn an inconsistent update, which makes the *difference between belief revision and updates* a little fuzzy. Nevertheless, such a difference is secondary in this research, as the general goal is to produce a robust semantics to represent knowledge with intuitive behaviour.

On the other hand, in this chapter there is a section devoted to show that the semantics coincides with other operators introduced in previous chapters, under certain restrictions. As a result, this semantics is more general and inherits the same theoretical basis and most of the properties from its predecessors.

Besides satisfying the principles proposed, this chapter illustrates through several

examples and transformations how to overcome problems occurring in alternative update approaches for ASP, in particular a *persistence situation*. One of the key transformations yields an abductive program out of a given updating pair of extended logic programs. Another transformation constructs a *preferred weak-constraint program* to compute *generalised answer sets* of the abductive program. Finally, as one main goals of Logic Programming, there is a section introducing a *functional prototype* from the declarative semantics version, that computes an *approximation* to the update answer sets of a given pair of programs by means of DLV's *weak-constraints models*, that fully correspond to the MSGAS's of a slight modification to the main update operation. Although its *cardinality criterion* of preference makes it applicable to slightly different kind of problems than the original operator, it allows to quickly implement a prototype naturally, in DATALOG$^{\vee,\omega}$, and gives enough background and technology to easily implement the *set-inclusion criterion*. The prototype is fully functional, helpful in exploration of new properties and applications, and it runs online with a standard browser interface. The section also includes an analysis of complexity of the corresponding processes that suggests a *polynomial time* in the case of *propositional theories*, imposed by the computing of EDLP's with weak constraints, which is not significantly affected by the update operation nor by the introduction of disjunction and weak constraints to ELP's.

In summary, the chapter has presented a more general framework that meets properties of iterated (*non-sequenced*) approaches and emphasises the logical contents of programs that represent knowledge. As a consequence, this semantics may overcome particular problems associated with *persistence* and *minimality of change* at the object level, as well as with a high dependence of syntax, giving a more general platform for more complex frameworks in knowledge representation. Finally, as an important component of Logic Programming, there is a solver prototype of this current semantics that closes the gap between theory and practice and opens a path with a solid component for further applications in management of knowledge systems.

In the courtroom of the conscience, a case is always in progress.

UNKNOWN

Chapter 9

Conclusions

Let us end this thesis by giving a summary of the pursued research, as well as an appraisal of the presented evidences and syntheses that support its claims. To begin with, Section 9.1 gives a general overview of the thesis, as well as an assessment of the original objectives. Next, Section 9.2 is a summarised list of major contributions that might be interesting to the research community.

9.1 Overview of the Thesis

The main objective of this thesis has been to explore whether an appropriate synthesis of well-known *principles and postulates* can be a suitable foundation for a simpler and stronger framework to update propositional theories in ASP, than earlier approaches. Once such a synthesis is achieved, the problem of updating knowledge bases turns more natural and intuitive, by overcoming unexpected circumstances that would bring the knowledge base to collapse. In consequence, a language with such a strong basis would lead to *correct knowledge and beliefs representation and reasoning*.

I have presented a general and simple semantics for updates of logic programs to support such a claim, whose main feature is a strong theoretical synthesis of general well-knonwn *principles and postulates*, and that it can be easily modified to meet particular needs of consistency and levels of abstraction.

Rather than setting up a very complex formulation to characterise other frameworks or to carry on building up syntactical ones with *brand-new principles*, this proposal starts with a study on theoretical bases to represent knowledge. In addition, there is a general study focused on the most relevant semantics in ASP to point out their features and limitations. As a result, this proposals overcomes major drawbacks of the other approaches, in a simple and robust semantics to reason and represent knowledge correctly. Finally, this work is also supported by an implemented *solver prototype* as an approximation and automatic testbed that makes the semantics more *accessible* (in a classroom, i.e.), and potential component for further more complex prototypes in administration of (toy?) knowledge systems, with precise properties.

9.2 Relevance of the Major Contributions

This section is a highlight of several major *contributions* of this dissertation that may be relevant to further study in the research community. Although they have an associated cardinal number, it does not necessarily reflect their impact or significance.

To begin with, the **first** contribution is a set of *challenging examples* to question, assess and compare existing semantics in a quick *survey* of the most relevant approaches that perform updates in ASP. Such a set may be useful as a preliminary "*benchmark*" to study properties of further semantics for updates, and has been carefully summarised into three tables for quick access in Appendix A.

The **second** contribution is an introduction to a general method of *relaxing knowledge bases* that allows to reason about conflicting new observations from the environment that might compromise the integrity of a (current) initial knowledge base. In addition to that, the method implies an initial specific set of *basic structural properties proposed* to be satisfied by any update approach in ASP. Finally, the approach may be tailored to particular inconsistency needs, as shown in subsequent chapters.

The **third** one is an introductory method to update *sequences of logic programs* that can meet the proposed properties, which may be tailored to model sequences of changes in more-complex frameworks with a different minimal-change criteria at the meta-level.

The number **four** is a *general framework to update* logic programs that can meet the proposed properties and most of the AGM-postulates, which may be customised to model changes on knowledge bases in more-complex frameworks at the .

The **fifth** contribution is a study on how to deal with *inconsistencies*, which is always relevant in administration of knowledge bases and provides methods to perform both belief updates and belief revisions that may be applied in different fields that require consistent knowledge bases.

A **sixth** contribution is a *characterisation of updates* with preferences of both cardinality and set-inclusion criteria. Such characterisation studies may be extended to further different *preference-criteria*, according to the particular problem to solve within diverse scenarios of minimal change and conflict resolution.

As a **seventh** contribution there are *automatic methods to translate* abductive programs into preference programs both in DLV's weak constraints and ODLP, which also may automate the process of finding their generalised answer sets. Such methods might be interesting in the implementation of particular needs of GAS's in other more-complex prototypes of diagnosis, debugging, preferences, etc.

Number **eighth**: a particular interpretation of *AGM-postulates in ASP* and a *semantics* that satisfies five out of six postulates that may be of interest to communities from both monotonic and non-monotonic reasoning.

Last but not least, another contribution is a *characterisation of updates* with two preference frameworks: *Ordered-disjunctive Logic Programming* and DLV's *Weak Constraints*, which leads to to *rapid-prototype*s frameworks of updates and make it more accessible to the research community, for the discovery of further properties and more-complex prototypes. The prototypes may also be easily tailored to particular needs of

the research community.

9.2.1 Update Benchmark

Chapter 4 and Chapter 5 present an analysis and compilation of the most relevant semantics for updates, as well as a set of *key examples* that jeopardise the integrity of the knowledge base in question, and that violate some common-sense intuitive principles in general. Such an analysis consists of pointing out some of their features and shortcomings than have motivated the work and scope of this thesis, by means of a set of counterexamples that suggest a "common *benchmark*" for further research on each proposal as well. Nevertheless, there is much work to be done to analyse the impact of such consideration in each particular case, as some of the proposals' main goals are more oriented to characterising different specific aspects to updating of knowledge bases.

The survey also includes an open question on other ways to tackle the problem of updating knowledge bases in other related frameworks of logic programming —i.e. *Well-founded Semantics* [Alferes et al., 2005, 1999; Van Gelder et al., 1991], *epistemic logic programming* [Zhang, 2003a, 2006]—, where they may find similar *counterintuitive problems* and which has been clearly out of the scope of this thesis. The work done in this thesis, however, confirms the suitability of AGM-theory and in particular KM′-postulates and other principles, general enough to be relevant even in other frameworks like *modal logic*, *linear logic*, *argumentation*, *nonmonotonic reasoning*, *default logic*, *planning*, etc.

9.2.2 Relaxation Technique

Another major contribution is a *relaxation technique*, to be used in the rest of the thesis, to an initial knowledge base. The method has been introduced and explored in Chapter 6, and then extended in Chapter 7 and Chapter 8 to meet particular needs. The technique is so simple that the process takes *linear time* to compute it, and makes an initial knowledge-base flexible so that it may cope with new possible-conflicting information from the environment. This means that new information has more *preference over background knowledge*, by following a proposed set of basic *structural properties* and *principles*.

In particular, weakening the original knowledge base consists in turning it into *incomplete knowledge*, so that one can complete it with new upcoming information by giving more priority to the latter and by preserving two kinds of *minimal-change principle* that have different implications on *persistence of beliefs*. The results of such method and properties have been encouraging and of great value to this entire thesis, to some related work of *updates in ASP*, and should be of interest to other more general fields of research like *nonmonotonic reasoning*, *commonsense reasoning*, *knowledge representation*, *constraint programming*, *argumentation*, *preferences*, *belief revision and updates*, *abduction*, and others.

9.2.3 Semantics for Belief Revision and Update

The previous background of relevant proposals and a relaxation technique leads to a general framework to update logic programs, which is another major contribution. For both historical and practical reasons, semantics for *updates* of logic programs are called so, without making any distinction to belief-revision operations. The difference may be quite subjective, although there is a key distinction when having an originally-inconsistent knowledge base or when originally-inconsistent information comes up from the environment, which are implications of a static or changing environment.

Chapters 6 and 7 take in these concerns and present a case study of managing two kinds of inconsistencies. Accordingly, the proposed method can be applied when *restoring consistency* from an *originally-inconsistent knowledge base*, as well as from an *originally-inconsistent update*. In addition to that, the same method can be applied when *strict consistency* is required in either case, which will lead more into the belief-update.

As a result, the semantics prove to comply with five out of six general postulates for belief revision of epistemic states, as well as with the full set of proposed basic structural properties. This result makes the semantics relevant to research communities from *nonmonotonic reasoning*, *commonsense reasoning*, *knowledge representation*, *argumentation*, *belief revision and updates*, *intelligent/rational/BDI-agents*, to mention a few.

9.2.4 Semantics for a Minimal Belief Change

Another main contribution of this research work, introduced in Chapter 4 and 6, and later developed in Chapter 7 and Chapter 8 are two case studies on how to deal with minimal-change principles, namely at the object-level and at the meta-level, with corresponding repercussions for *persistency of beliefs*. Chapter 8 comprises a preliminary empirical evaluation and comparison between the two approaches and leaves room to customise more than one minimality-criteria from the literature.

As a result, Chapter 7 consists of a semantics that performs updates at the meta-level with a minimal-*cardinality criterion*. In contrast, Chapter 8 comprises an object-level semantics both with a *minimal-cardinality* and *minimal-set-inclusion* criteria, besides a formal *equivalence-relation* between the two proposed semantics. Consequently, the method leaves room to easily implement those alternatives from the literature and more.

These minimal-change results should be interesting and useful for different research communities like *nonmonotonic reasoning*, *commonsense reasoning*, *knowledge representation*, *constraint programming*, *argumentation*, *preferences and qualitative decision making*, *belief revision and updates*, and others.

9.2.5 A Semantics Independent from Syntax

A major result out of the survey of other approaches, two properties proved to be cornerstones to overcome their counterintuitive behaviour of most of them: *weak irrelevance of syntax* and *strong consistency*.

While the majority of alternative proposals here studied are based on syntactic methods to perform updates, these two principles suggest a opposite method to agree with general postulates.

Preliminary versions of syntax-based approaches started to show unforeseen circumstances that caused counterintuitive behaviour. As a result, some improvements emerged to ad-hoc solutions that might not guarantee further unforeseen situations.

The proposed semantics presented along this thesis are model-based approaches in the sense that both the change policies and the resulting knowledge bases depend on *semantical-logical contents* of the updating knowledge bases, rather than depending upon the particular way to write them up. This resulted in an update semantics that meets the proposed structural properties as well as most of the AGM-postulates, besides overcoming the counterintuitive behaviour of the other alternatives.

The obvious repercussions of such a result lies in the potential *autonomy* of a computer system that is robust to contradictory situations. A robust system just described should be interesting to research communities of *preferences and qualitative reasoning*, *automated reasoning*, *optimisation*, *logic programming*, *nonmonotonic reasoning*, *commonsense reasoning*, *knowledge representation*, *constraint programming*, *belief revision and updates*, and more.

9.2.6 Preference Characterisations

Last but not least, another main contribution is the characterisations of updates both with *ordered disjunctions* and with *weak constraints*. These characterisations have been introduced from Chapter 6 to Chapter 8, and allow one to easily and quickly implement the proposed approaches without worrying too much about *imperative* or *procedural* details, for the declarative nature of the proposed programming tools. Likewise, the characterisations include a formal analysis of *complexity* that shows no significant additional throughput to the process of computing "regular" answer sets, despite the fact that the study did not pursue any *efficient implementation* for being beyond the intended thesis scope.

The proposed characterisations in ODLP and DLV's weak constraints allow one to express given updates, to an initial knowledge base, into a single corresponding DATALOG$^{\times}$ or DATALOG$^{\vee,\omega}$-program; as well as to compute both minimal generalised answer sets and MSGAS's in general. Such characterisations open up the possibility to reason updates within *preference frameworks* and their respective properties. An immediate result, for instance, is the *implemented prototypes* that are a contribution from this research topic. Such functional prototypes are possible when ODLP and weak-constraints solvers exist, like PSmodels and DLV's weak-constraints, respectively,

and allow one to choose between the two preference solvers. As a result, this thesis includes formal translations and an implemented prototype for *update sequences*[1] in DATALOG$^{\vee,\omega}$; another one for *iterated update*[2] in DATALOG$^{\vee,\omega}$; as well as a preliminary study[3] to use DATALOG$^{\times}$. All of them are widely available and running online.

Besides providing an automatic testbed for properties study, for the classroom, and for other more-complex prototypes, a great advantage of online applications is that they needn't downloaded or installed and they are easy to use for their standard web interface. In consequence, the proposed approaches widespread available online[4] are also easy to distribute, and should be of great interest to research communities of `PSmodels`, `DLV`, *preferences and qualitative reasoning*, *automated reasoning*, *optimisation*, *logic programming*, *commonsense reasoning*, *knowledge representation*, *argumentation*, and more.

[1] `http://www.in.tu-clausthal.de/~guadarrama/updates/seqs.html`
[2] `http://www.in.tu-clausthal.de/~guadarrama/updates/o.html`
[3] `http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html`
[4] `http://www2.in.tu-clausthal.de/~guadarrama/updates/`

Bibliography

Alchourrón, C. E., Gärdenfors, P., and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic 50,* 2 (June), 510–530. 4, 7, 15, 20, 21, 43, 136

Alferes, J. J., Banti, F., Brogi, A., and Hitzler, P. 2005. The well supported semantics for multidimensional dynamic logic programs. In *Logic Programming and Nonmonotonic Reasoning*, C. Baral, G. Greco, N. Leone, and G. Terracina, Eds. LNCS, vol. 3662/2005. Springer Berlin/Heidelberg, Diamante, Italy, 356–368. 49, 50, 53

Alferes, J. J., Banti, F., Brogi, A., and Leite, J. A. 2005. The refined extension principle for semantics of dynamic logic programming. *Studia Logica 79,* 1, 7–32. 7, 10, 44, 48, 49, 50, 51, 53, 95, 96, 109, 110, 135, 136, 138, 139, 141, 148, 169, 188

Alferes, J. J., Leite, J. A., Pereira, L. M., Przymusinska, H., and Przymusinski, T. C. 1999. Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming 45,* 1–3, 43–70. 9, 43, 44, 49, 52, 53, 57, 58, 59, 88, 96, 97, 109, 136, 137, 138, 139, 169

Alferes, J. J., Leite, J. A., Pereira, L. M., Przymusinski, T. C., and Przymusinska, H. 1998. Dynamic logic programming. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, A. Cohn, L. Schubert, and S. Shapiro, Eds. Morgan Kaufmann Publishers, San Francisco, 98–111. 49

Alferes, J. J., Pereira, L. M., Przymusinska, H., and Przymusinski, T. C. 2002. LUPS —A language for updating logic programs. *Artificial Intelligence 138,* 1–2 (June), 87–116. 49, 189

Balduccini, M. and Gelfond, M. 2003. Logic programs with consistency-restoring rules. In *Proceedings of the AAAI Spring 2003 Symposium.* AAAI Press, Palo Alto, California, 9–18. 38, 39, 40, 95, 104

Benferhat, S., Kaci, S., Berre, D. L., and Williams, M.-A. 2004. Weakening conflicting information for iterated revision and knowledge integration. *Artificial Intelligence 153,* 1–2, 339–371. 189

Brewka, G. 2001. Declarative representation of revision strategies. *Journal of Applied Non-classical Logics 11,* 1–2, 151–167. 136

Brewka, G. 2002. Logic programming with ordered disjunction. In *Proceedings of the 18th National Conference on Artificial Intelligence, AAAI-2002.* Morgan Kaufmann, Edmonton, Alberta, Canada. 35, 95, 101, 126, 156

Brewka, G. and Dix, J. 1998. Knowledge representation with logic programs. In *Logic Programming and Knowledge Representation*, J. Dix, L. Pereira, and T. Przymusinski, Eds. LNAI 1471. Springer, Berlin, 1–55. Full version will appear as Chapter 6 in *Handbook of Philosophical Logic*, 2nd edition (2005), Volume 6, Methodologies. 29

Brewka, G., Niemelä, I., and Syrjänen, T. 2002. Implementing ordered disjunction using answer set solvers for normal programs. In *Logics in Artificial Intelligence, 8th European Conference, JELIA*, G. Goos, J. Hartmanis, and J. v. Leeuwen, Eds. LNCS, vol. 2424/2002. Springer-Verlag Berlin/Heidelberg, 444–456. 38, 95, 104, 188

Brewka, G., Niemelä, I., and Syrjänen, T. 2004. Logic programs with ordered disjunction. *Computational Intelligence 20,* 2, 333–357. 35, 36, 37, 38, 101, 102, 103, 188, 197

Brouwer, L. E. J. 1907. On the foundations of mathematics. Ph.D. thesis, School of Mathematics, Amsterdam. 3, 16

Buccafurri, F., Leone, N., and Rullo, P. 1997. Strong and weak constraints in disjunctive datalog. In *Logic Programming and Nonmonotonic Reasoning.* LNCS, vol. 1265/1997. Springer, 2–17. 162

Buccafurri, F., Leone, N., and Rullo, P. 2000. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering 12,* 5, 845–860. 33, 35, 41, 162

Cai, J.-Y. 2003. *Lectures in Computational Complexity.* unpublished, Department of Computer Science, University of Wisconsin. 40

Calimeri, F., Dell'Armi, T., Eiter, T., Faber, W., Gottlob, G., Ianni, G., Ielpa, G., Koch, C., Leone, N., Perri, S., Pfeifer, G., and Polleres, A. 2002. The DLV System. In *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA)*, S. Flesca and G. Ianni, Eds. Springer, Cosenza, Italy. 7, 30, 155

Crescini, V. F. and Zhang, Y. 2005. Policy updater: A system for dynamic access control. *International Journal of Information Security 5,* 3, 145–165. 30, 65, 189

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. 1999. Complexity and expressive power of logic programming. Tech. Rep. INFSYS RR-1843-99-05, TU Wien, Viena, Austria. February. 162

Darwiche, A. and Pearl, J. 1994. On the logic of iterated belief revision. In *Proceedings of the fifth Conference on Theoretical Aspects of Reasoning about Knowledge*, R. Fagin, Ed. Morgan Kaufmann, Pacific Grove, CA, 5–23. ix, 4, 21, 22, 25, 151

Dix, J. 1995a. A classification-theory of semantics of normal logic programs: I. strong properties. *Fundamenta Informaticae XXII(3)*, 227–255. 7

Dix, J. 1995b. A classification theory of semantics of normal logic programs: II. weak properties. *Fundamenta Informaticae XXII(3)*, 257–288. 7, 109

Eiter, T., Faber, W., Leone, N., and Pfeifer, G. 2002. Computing preferred answer sets by meta-interpretation in answer set programming. Tech. Rep. INFSYS RR-1843-02-01, Vienna University of Technology. 44

Eiter, T., Fink, M., Sabbatini, G., and Tompits, H. 2000a. Considerations on updates of logic programs. In *Logics in Artificial Intelligence, European Workshop, JELIA 2000*, M. Ojeda-Aciego, I. P. de Guzmán, G. Brewka, and L. Moniz Pereira, Eds. Springer Verlag, Malaga, Spain, 2–20. 43, 96, 109, 136

Eiter, T., Fink, M., Sabbatini, G., and Tompits, H. 2000b. On updates of logic programs: Semantics and properties. Tech. Rep. INFSYS RR-1843-00-08, TU Wien, Institute für Informationssysteme. 43, 44, 96

Eiter, T., Fink, M., Sabbatini, G., and Tompits, H. 2001. A framework for declarative update specifications in logic programs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI*, B. Nebel, Ed. Vol. I. Morgan Kaufmann, Seattle, Washington, 649–654. 7, 43, 96

Eiter, T., Fink, M., Sabbatini, G., and Tompits, H. 2002. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming 2,* 6, 711–767. 7, 8, 9, 24, 43, 44, 45, 47, 48, 49, 53, 59, 80, 81, 87, 91, 96, 97, 99, 109, 118, 119, 120, 132, 135, 136, 148, 187, 193, 198

Eiter, T., Fink, M., Sabbatini, G., and Tompits, H. 2005. Reasoning about evolving nonmonotonic knowledge bases. *ACM Transactions on Computational Logic 6,* 2, 389–440. 7, 30, 44, 96, 136

Fagin, R. 1995. *Reasoning about Knowledge.* The MIT Press, Cambridge, Massachusetts, USA. 43

Fagin, R., Kuper, G. M., Ullman, J. D., and Vardi, M. Y. 1986. Updating logical databases. *Advances in Computing Research 3*, 1–18. 43

Fagin, R., Ullman, J. D., and Vardi, M. Y. 1983. On the semantics of updates in databases. In *PODS '83: Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*. ACM Press, New York, NY, USA, 352–365. 43

Gärdenfors, P. and Makinson, D. 1988. Revisions of knowledge systems using epistemic entrenchment. In *Proceedings of the second Conference on Theoretical Aspects of Reasoning about Knowledge*, M. Y. Vardi, Ed. Morgan Kaufmann, Pacific Grove, CA. 20

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium ICLP/SLP*, R. A. Kowalski and K. A. Bowen, Eds. MIT Press, Seattle, Washington. 6, 27, 28, 43, 187

Guadarrama, J. C. 2007a. Implementing knowledge update sequences. In *MICAI 2007: Advances in Artificial Intelligence*, A. Gelbukh and A. Kuri Morales, Eds. LNCS, vol. 4827. Springer-Verlag, Aguascalientes, Mexico, 260–270. 13, 14, 188

Guadarrama, J. C. 2007b. Implementing knowledge update sequences. In *MICAI 2007: Advances in Artificial Intelligence*, A. Gelbukh and A. Kuri Morales, Eds. LNCS, vol. 4827. Springer-Verlag Berlin/Heidelberg, Aguascalientes, Mexico, 1–8. 188

Guadarrama, J. C. 2007c. Maintaining knowledge bases at the object level. In *Special Session of the 6th International MICAI Conference*, A. Gelbukh and A. F. Kuri Morales, Eds. IEEE Computer Society, Aguascalientes, Mexico, 3–13. ISBN: 978-0-7695-3124-3. 14, 188

Guadarrama, J. C. 2007d. A road map of updating in ASP. ISSN: 1860-8477 IfI-07-16, Institute für Informatik, TU-Clausthal, Clausthal, Germany. December. 13

Guadarrama, J. C. 2008a. AGM postulates in answer sets. In *LANMR'08 Fourth Latin American Workshop on Non-monotonic Reasoning*, M. Osorio and I. Olmos, Eds. ISSN 1613-0073, vol. 408. CEUR. 14

Guadarrama, J. C. 2008b. A system to maintain knowledge bases of intelligent agents in DLV's weak constraints. In *Artificial Intelligence and Soft Computing*, A. P. del Pobil, Ed. Number 628. IASTED/ACTA Press, Palma de Mallorca, Spain. ISBN: 978-0-88986-755-0. 14

Guadarrama, J. C. 2008c. Update operation in ASP revisited. ISSN: 1860-8477 IfI-08-12, Institute für Informatik, TU-Clausthal, Clausthal, Germany. December. 13, 95, 109

Guadarrama, J. C. 2009. Maintaining sequences of knowledge bases in ASP. ISSN: 1860-8477 IfI-09-11, Institute für Informatik, TU-Clausthal, Germany. November. 13

Guadarrama, J. C., Arrazola, J., and Osorio, M. 2002. Making belief revision with LUPS. In *XI International Conference on Computing*, J. H. S. Azuela and G. A. Figueroa, Eds. CIC-IPN, PO Box 75-476 Col. Nueva Industrial Vallejo 07738, México, D.F. 18

Guadarrama, J. C., Dix, J., and Osorio, M. 2006. Update sequences in Generalised Answer Set Programming based on structural properties. In *Special Session of the 5th International MICAI Conference*, P. Kellenberger, Ed. IEEE Computer Society, Mexico City, Mexico, 32–41. ISBN: 0-7695-2722-1. 13, 59, 188

Guadarrama, J. C., Dix, J., Osorio, M., and Zacarías, F. 2005. Updates in Answer Set Programming based on structural properties. In *7th International Symposium on Logical Formalizations of Commonsense Reasoning*, S. McIlraith, P. Peppas, and M. Thielscher, Eds. Fakultät Informatik, ISSN 1430-211X, Corfu, Greece, 213–219. 13, 14, 39, 109, 188

Guadarrama, J. C. and Osorio, M. 2002. Exploring belief revision with LUPS. In *Avances en Inteligencia Artificial*, J. H. S. Azuela and G. A. Figueroa, Eds. CIC-IPN and SMIA, PO Box 75-476 Col. Nueva Industrial Vallejo 07738, México, D.F. 18

Guadarrama, J. C. and Osorio, M. J. 2003. Towards modelling an intelligent calendar agent with LUPS. In *Applied Informatics*, M. H. Hamza, Ed. IASTED/ACTA Press, Insbruck, Austria, 60–65. 18

Heymans, S. 2006. Decidable open answer set programming. Ph.D. thesis, Vrije Universiteit Brussel, Faculty of Science, Department of Computer Science, Theoretical Computer Science. 162

Inoue, K. and Sakama, C. 1995. Abductive framework for nonmonotonic theory change. In *the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*. Morgan Kaufmann Publishers, Montreal, Canada, 204–210. 55

Inoue, K. and Sakama, C. 2004. Equivalence of logic programs under updates. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA'04)*, J. J. Alferes and J. A. Leite, Eds. LNAI, vol. 3229. Springer-Verlag, Lisbon, Portugal, 174–186. 30, 32

Johnson, D. S. 1990. A catalog of complexity classes. In *Handbook of theoretical computer science: algorithms and complexity*. Vol. A. MIT Press, Cambridge, MA, USA, 67–161. 27, 40, 41

Kakas, A., Kowalski, R., and Toni, F. 1998. The role of abduction in logic programming. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Eds. Vol. 5, Logic programming. Oxford University Press, Oxford, UK, 235–324. 54, 55

Kakas, A. C. and Mancarella, P. 1990. Generalized Stable Models: A semantics for abduction. In *ECAI*. Stockholm, Sweden, 385–391. 38, 39, 54, 95

Katsuno, H. and Mendelzon, A. O. 1989. A unified view of propositional knowledge base updates. In *the 11th International Joint Conference on Artificial Intelligence, IJCAI-89*, N. S. Sridharan, Ed. Morgan Kaufmann, Detroit, Michigan, USA, 1413–1419. 43

Katsuno, H. and Mendelzon, A. O. 1991a. On the difference between updating a knowledge base and revising it. In *KR'91*. Morgan Kaufmann Publishers, Cambridge, Massachusetts, USA. ix, 5, 23, 24, 25, 125, 140

Katsuno, H. and Mendelzon, A. O. 1991b. Propositional knowledge base revision and minimal change. *Artificial Intelligence 52,* 3, 263–294. ix, 4, 7, 21, 22, 43, 99

Kraus, S., Lehmann, D., and Magidor, M. 1990. Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence 44,* 1, 167–207. 43

Lehmann, D. 1992. Plausibility Logic. In *Computer Science Logic, 5th Workshop, CSL 91, Berne, Switzerland*, E. Börger, G. Jäger, H. Kleine-Büning, and M. M. Richter, Eds. LNCS 626. Springer, Berlin, 227–241. 43

Lehmann, D. and Magidor, M. 1992. What does a conditional knowledge base entail? *Artificial Intelligence 55*, 1–60. 43

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562. 7, 30, 33, 34, 35, 111, 126, 127, 128, 132, 155, 156, 157, 162

Lifschitz, V., Pearce, D., and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2,* 4 (October), 526–541. 7, 17, 30, 31, 32

Lifschitz, V. and Woo, T. 1992. Answer sets in general non-monotonic reasoning (preliminary report). In *Third International Conference on Principles of Knowledge Representation and Reasoning*. Morgan-Kaufmann, Cambridge, Massachusetts, USA, 603–614. 6, 27, 187

Lloyd, J. W. 1987. *Foundations of Logic Programming*. Springer, Berlin. 2nd edition. 27

Makinson, D. 1988. General theory of cumulative inference. In *Non-Monotonic Reasoning, 2nd International Workshop*. Springer, Grassau, Federal Republic of Germany, 1–18. 43

Makinson, D. 1994. General Patterns in Nonmonotonic Reasoning. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Nonmonotonic and Uncertain Reasoning*, D. Gabbay, C. J. Hogger, and J. A. Robinson, Eds. Vol. 3. Oxford University Press, Oxford, UK, Chapter 3, 35–110. 43

Marek, V. W. and Truszczynski, M. 1994. Revision specifications by means of programs. In *Logics in Artificial Intelligence, European Workshop, JELIA*. Springer, York, UK, 122–136. 74

Marek, V. W. and Truszczynski, M. 1998. Revision programming. *Theoretical Computer Science 190,* 2 (January), 241–277. 74

Meyer, A. R. and Stockmeyer, L. J. 1972. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT '72: Proceedings of the 13th Annual Symposium on Switching and Automata Theory (swat 1972)*. IEEE Computer Society, Washington, DC, USA, 125–129. 40

Mints, G. 2000. *A Short Introduction to Intuitionistic Logic*. Kluwer Academic/Plenum Publishers, New York, NY, USA. 3, 16

Moschovakis, J. 1999. Intuitionistic logic. In *The Stanford Encyclopedia of Philosophy*, E. Zalta, Ed. The Metaphysics Research Lab, Stanford University, Stanford, CA 94305-4115. 16, 17

Nelson, D. 1949. Constructible falsity. *Journal of Symbolic Logic 14,* 1, 16–26. 31

Niemela, I. and Simons, P. 1997. Smodels —an implementation of the Stable Model and Well-Founded Semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*. Lecture Notes in Artificial Intelligence (LNCS), vol. 1265. Springer, Dagstuhl Castle, Germany, 420–429. 7, 30, 38, 104

Ortiz, M. and Osorio, M. 2005. Nelson's strong negation, safe beliefs and the answer set semantics. In *Answer Set Programming*. 136

Osorio, M. and Cuevas, V. 2007. Updates in answer set programming: An approach based on basic structural properties. *Journal of Theory and Practice of Logic Programming 7,* 4, 451–479. 8, 9, 31, 32, 81, 136, 137

Osorio, M., Navarro, J. A., and Arrazola, J. 2001. Equivalence in answer set programming. In *Logic Based Program Synthesis and Transformation:* 11^{th} *International Workshop; selected papers / LOPSTR 2001, Paphos, Cyprus*, A. Pettorossi, Ed. LNCS 2372. Springer-Verlag Berlin, Paphos, Cyprus, 57–75. 31, 49, 86, 110, 142

Osorio, M., Navarro, J. A., and Arrazola, J. 2004. Applications of intuitionistic logic in answer set programming. *Theory and Practice of Logic Programming 4,* 3, 325–354. 27, 28

Osorio, M., Navarro, J. A., and Arrazola, J. 2005. Safe beliefs for propositional theories. *Annals of Pure and Applied Logic 134,* 1, 63–82. 18

Osorio, M., Ortiz, M., and Zepeda, C. 2004. Using cr-rules for evacuation planning. In *Workshop Proceedings of Deduction and Reasoning Techniques*. IEEE Computer Society, Puebla, Mexico. 38, 104, 105

Osorio, M. and Zacarías, F. 2003. Irrelevance of syntax in updating answer set programs. In *Proceedings of the Fourth Mexican International Conference on Computer Science (ENC' 03) In Workshop on Logic and Agents*. IEEE Computer Society, Apizaco, Mexico. 43, 44, 95, 96

Osorio, M. and Zacarías, F. 2004. On updates of logic programs: A properties-based approach. In *FoIKS*. Springer, Wilhelminenburg Castle, Austria, 231–241. 81, 95, 96, 109, 135

Pearce, D. 1999a. From here to there: Stable negation in logic programming. In *What Is Negation?*, D. M. Gabbay and H. Wansing, Eds. Applied Logic Series, vol. 13. Kluwer Academic Publishers, Dordrecht, The Netherlands, 161–181. 6, 17, 18, 19, 27, 28, 31, 136

Pearce, D. 1999b. Stable inference as intuitionistic validity. *The Journal of Logic Programming 38*, 79–91. 27, 31

Poole, D. 1988. A logical framework for default reasoning. *Artificial Intelligence 36,* 1, 27–47. 54

Safra, J. E. and Yeshua, I. 2002. Logic systems. In *The New Encyclopaedia Britannica: macropaedia*, 15th ed. Encyclopedia Britannica, Inc., Chicago, Il, USA. 15

Sakama, C. and Inoue, K. 1999. Updating extended logic programs through abduction. In *LPNMR '99: Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning*, M. Gelfond, N. Leone, and G. Pfeifer, Eds. LNCS, vol. 1730. Springer-Verlag, El Paso, Texas, USA, 147–161. 43, 53, 96

Sakama, C. and Inoue, K. 2003. An abductive framework for computing knowledge base updates. *Theory and Practice of Logic Programming 3,* 6, 671–715. 9, 43, 52, 53, 54, 55, 56, 57, 58, 59, 60, 64, 65, 88, 92, 96, 109, 118, 125, 135, 136, 137, 138, 139, 140, 149, 150, 151, 165

Shankar, N. 1997. *Metamathematics, Machines and Gödel's Proof*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK. 27

Van Gelder, A., Ross, K. A., and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38,* 1 (January). 7, 169

Winslett, M. 1990. *Updating logical databases*. Cambridge University Press, New York, NY, USA. 43

Zhang, Y. 1995. On propositional knowledge base updates. *Australian Journal of Intelligent Information Processing Systems 2*, 20–29. 7, 43, 65

Zhang, Y. 2001. The complexity of logic program updates. In *AI '01: Proceedings of the 14th Australian Joint Conference on Artificial Intelligence*. Springer-Verlag, London, UK, 631–642. 43

Zhang, Y. 2003a. Minimal change and maximal coherence for epistemic logic program updates. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, Acapulco, Mexico, 112–120. 81, 169

Zhang, Y. 2003b. Two results for prioritized logic programming. *Theory and Practice of Logic Programming 3,* 2 (March), 223–242. 65, 67, 68, 70

Zhang, Y. 2006. Logic program-based updates. *ACM Transactions on Computational Logic 7,* 3 (July), 421–472. 7, 8, 9, 43, 45, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 77, 78, 80, 81, 90, 91, 96, 138, 169, 188, 193, 194, 195, 196, 197

Zhang, Y. and Foo, N. 2005. A unified framework for representing logic program updates. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press / The MIT Press, Pittsburgh, Pennsylvania, USA, 707–713. 66, 90, 96, 109, 135, 136

Zhang, Y. and Foo, N. Y. 1997. Answer sets for prioritized logic programs. In *ILPS '97: Proceedings of the 1997 international symposium on Logic programming*. MIT Press, Cambridge, MA, USA, 69–83. 67

Zhang, Y. and Foo, N. Y. 1998. Updating logic programs. In *ECAI'98 13th European Conference on Artificial Intelligence*, H. Prade, Ed. John Wiley & Sons, Ltd., Brighton, UK, 403–407. 53, 65

Appendix A

Summary of Properties

Following there is a summary of the fundamental properties I propose an update semantics ought to meet. A tick like "✓" means the corresponding semantics meets the property, while "✗" stands for the converse with one or more existing counterexamples and/or definition. Lastly, when there exists no known proof nor counterexample available, the corresponding symbol is just a blank. In particular, Table A shows a list of properties and those semantics that meet or not the corresponding property. Next, Table A shows the properties from Table A in an affirmative fashion. Finally, Table A shows the properties from Table A in a negative fashion.

As discussed in Section 2.3, recall that AGM-postulates (PK∗2), (PK∗4) and (PK∗6) in Table 8.1 correspond to KM′-postulates (R ∘ 1), (R ∘ 2) and (R ∘ 4), respectively, from Table 2.5.

Operator: / From Section(s):	$\oplus_R$ 4.2	$\diamond_{\mathsf{SI}}$ 4.3	$\oplus_2$ 4.5	$\oplus_3$ 4.5	$\diamond_{\mathsf{Z}}$ 4.4	$\triangleleft$ 4.1,4.5	$\oslash$ 6.1	$\otimes$ 7.2	$\otimes'_o$ 8.4.2	$\otimes'$ 7.4	$\otimes_o$ 8.2
Augt. Upd. Sec. 6.2			✓	×		×	✓	✓	✓	✓	✓
Consy. Res. Sec. 7.3.1		✓			×		✓	✓	✓	✓	✓
Idempotence Sec. 6.2			✓	✓		✓	✓	✓	✓	✓	✓
Inertia Sec. 6.2		✓					✓	✓	✓	✓	✓
Initialitation Sec. 6.2		×	✓	✓		✓	✓	✓	✓	×	✓
Integrity Constraints		✓	×	×	×	×	×	✓	✓	✓	✓
Object level Sec. 5.2	×	✓	×	×	✓	×	×	×	✓	×	✓
Conflicting I. Sec. 5.3	×	×	×	×	✓	×	×	×	×	×	×
$\mathcal{L}_{\mathsf{ELP}}$Sec. 3.2	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\mathcal{L}_{\mathsf{GLP}}$Sec. 4.2	✓	×	×	×	×	×	×	×	×	×	×
Min.C. Sec. 1.4.1		✓			✓		✓	✓	✓	✓	✓
Multiple Updates	✓	✓	×	×	×	✓	×	✓	✓	✓	✓
SC Sec. 6.2	×		×	×	×	×	✓	✓	✓	✓	✓
WC Sec. 7.3.2							✓	✓	✓	✓	✓
WIS Sec. 6.2	×			×		×	✓	✓	✓	✓	✓
WNI Sec. 6.2			×	✓		✓	✓	✓	✓	✓	✓
(R ∘ 1) Sec. 2.3			✓	✓		✓	✓	✓	✓	×	✓
(R ∘ 2) Sec. 2.3			×	×		×	✓	✓	✓	✓	✓
(R ∘ 3) Sec. 2.3							✓	✓	✓	✓	✓
(R ∘ 4) Sec. 2.3			✓	×		×	✓	✓	✓	✓	✓
(R ∘ 5) Sec. 2.3				✓		✓	×				×
(R ∘ 6) Sec. 2.3				×		×	✓				✓
(PK ∗ 1) Sec. 8.1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(PK ∗ 3) Sec. 8.1			✓	✓		✓	✓	✓	✓	✓	✓
(PK ∗ 5) Sec. 8.1			✓	✓		✓	✓	✓	✓	×	✓
Assoc. Sec. 8.3.5							×				✓
Distrib. Sec. 8.3.5							✓				✓
W.Comm. Sec. 8.3.5							✓				✓
Epist.St. Sec. 2.4	×		×	×	×	×	×	×	✓	×	✓

Table A.1: Summary of General Properties

Operator: / From Section(s):	$\oplus_R$ 4.2	$\diamond_{\mathsf{SI}}$ 4.3	$\oplus_2$ 4.5	$\oplus_3$ 4.5	$\diamond_{\mathsf{Z}}$ 4.4	$\triangleleft$ 4.1,4.5	$\oslash$ 6.1	$\otimes$ 7.2	$\otimes'_o$ 8.4.2	$\otimes'$ 7.4	$\otimes_o$ 8.2
Augt. Upd. Sec. 6.2			✓				✓	✓	✓	✓	✓
Consy. Res. Sec. 7.3.1		✓					✓	✓	✓	✓	✓
Idempotence Sec. 6.2			✓	✓		✓	✓	✓	✓	✓	✓
Inertia Sec. 6.2		✓					✓	✓	✓	✓	✓
Initialitation Sec. 6.2			✓	✓		✓	✓	✓	✓		✓
Integrity Constraints		✓						✓	✓	✓	✓
Object level Sec. 5.2		✓			✓				✓		✓
Conflicting I. Sec. 5.3					✓						
$\mathcal{L}_{\mathsf{ELP}}$Sec. 3.2		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\mathcal{L}_{\mathsf{GLP}}$Sec. 4.2	✓										
Min.C. Sec. 1.4.1		✓			✓		✓	✓	✓	✓	✓
Multiple Updates	✓	✓				✓		✓	✓	✓	✓
SC Sec. 6.2							✓	✓	✓	✓	✓
WC Sec. 7.3.2							✓	✓	✓	✓	✓
WIS Sec. 6.2							✓	✓	✓	✓	✓
WNI Sec. 6.2				✓		✓	✓	✓	✓	✓	✓
(R $\circ$ 1) Sec. 2.3			✓	✓		✓	✓	✓	✓		✓
(R $\circ$ 2) Sec. 2.3							✓	✓	✓	✓	✓
(R $\circ$ 3) Sec. 2.3							✓	✓	✓	✓	✓
(R $\circ$ 4) Sec. 2.3			✓				✓	✓	✓	✓	✓
(R $\circ$ 5) Sec. 2.3				✓		✓					
(R $\circ$ 6) Sec. 2.3							✓				✓
(PK $*$ 1) Sec. 8.1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(PK $*$ 3) Sec. 8.1			✓	✓		✓	✓	✓	✓	✓	✓
(PK $*$ 5) Sec. 8.1			✓	✓		✓	✓	✓	✓		✓
Assoc. Sec. 8.3.5											✓
Distrib. Sec. 8.3.5							✓				✓
W.Comm. Sec. 8.3.5							✓				✓
Epist.St. Sec. 2.4									✓		✓

Table A.2: Summary of General Properties

Operator: / From Section(s):	$\oplus_R$ 4.2	$\diamond_{\mathsf{SI}}$ 4.3	$\oplus_2$ 4.5	$\oplus_3$ 4.5	$\diamond_{\mathsf{Z}}$ 4.4	$\triangleleft$ 4.1,4.5	$\oslash$ 6.1	$\otimes$ 7.2	$\otimes'_o$ 8.4.2	$\otimes'$ 7.4	$\otimes_o$ 8.2
Augt. Upd. Sec. 6.2				×		×					
Consy. Res. Sec. 7.3.1					×						
Idempotence Sec. 6.2											
Inertia Sec. 6.2											
Initialitation Sec. 6.2		×								×	
Integrity Constraints			×	×	×	×	×				
Object level Sec. 5.2	×		×	×		×	×	×		×	
Conflicting I. Sec. 5.3	×	×	×	×		×	×	×	×	×	×
$\mathcal{L}_{\mathsf{ELP}}$Sec. 3.2	×										
$\mathcal{L}_{\mathsf{GLP}}$Sec. 4.2		×	×	×	×	×	×	×	×	×	×
Min.C. Sec. 1.4.1											
Multiple Updates			×	×	×		×				
SC Sec. 6.2	×		×	×	×	×					
WC Sec. 7.3.2											
WIS Sec. 6.2	×			×		×					
WNI Sec. 6.2			×								
(R $\circ$ 1) Sec. 2.3										×	
(R $\circ$ 2) Sec. 2.3			×	×		×					
(R $\circ$ 3) Sec. 2.3											
(R $\circ$ 4) Sec. 2.3				×		×					
(R $\circ$ 5) Sec. 2.3							×				×
(R $\circ$ 6) Sec. 2.3				×		×					
(PK $*$ 1) Sec. 8.1											
(PK $*$ 3) Sec. 8.1											
(PK $*$ 5) Sec. 8.1										×	
Assoc. Sec. 8.3.5							×				
Distrib. Sec. 8.3.5											
W.Comm. Sec. 8.3.5											
Epist.St. Sec. 2.4	×		×	×	×	×	×	×		×	

Table A.3: Summary of General Properties

Appendix B

Software-support Summary

B.1 Solvers

Following, this section consists of a list of solvers and applications, in no particular order, with their respective links to implementations. Some of them were adapted by myself in order to have a classical web-interface to run the *original untouched engine* online, with the obvious advantages.

`SMODELS` [Gelfond and Lifschitz, 1988]

- Section 3.2
- Original source at
  `http://www.tcs.hut.fi/Software/smodels/`
- Original engine with my web-interface at
  `http://www2.in.tu-clausthal.de/~guadarrama/updates/smodels.html`

`DLV` [Lifschitz and Woo, 1992]

- Section 3.2
- Original source at
  `http://www.dbai.tuwien.ac.at/proj/dlv/`
- Original engine with my web-interface at
  `http://www2.in.tu-clausthal.de/~guadarrama/updates/dlv.html`

◁ [Eiter et al., 2002]

- Section 4.1

- Original source at
  `http://www.kr.tuwien.ac.at/staff/giuliana/project.html#Download`
- Original engine with my web-interface at
  `http://www2.in.tu-clausthal.de/~guadarrama/updates/upd.html`

$\oplus_R$ [Alferes et al., 2005]

- Section 4.2
- Original source at
  `http://centria.di.fct.unl.pt/~banti/FedericoBantiHomepage/refdlp.htm`
- Original source previous to their *refined principle* at
  `http://centria.di.fct.unl.pt/~jja/updates/dlp.html`

$\mathbf{U}_{\diamond_\mathbf{Z}}(\Pi_0, \Pi_1)$ [Zhang, 2006]

- Section 4.4
- `www.cit.uws.edu.au/~yan/plps.html` —interpreter for PLP's
- `www.cs.uni-potsdam.de/~torsten/plp/` —Alternative compiler for PLP's running on Prolog

$\oslash$ [Guadarrama et al., 2005]

- Chapter 6
- `http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html`

$\otimes$ [Guadarrama, 2007a,b; Guadarrama et al., 2006]

- Chapter 7
- `http://www.in.tu-clausthal.de/~guadarrama/updates/seqs.html`

$\otimes_o$ [Guadarrama, 2007c]

- Chapter 7
- `http://www.in.tu-clausthal.de/~guadarrama/updates/o.html`

ODLP [Brewka et al., 2002, 2004]

- Section 3.5
- Original source at
  `http://www.tcs.hut.fi/Software/smodels/priority/`
- Original source previous to their *refined principle* at
  `www.in.tu-clausthal.de/~guadarrama/updates/psmodels.html`

AGM [Benferhat et al., 2004]

- Section 2.3
- `www.uni-koblenz.de/ag-ki/LP/system.SATEN.html`

B.2 Applications

LUPS [Alferes et al., 2002]

- Section 4.2
- `http://centria.di.fct.unl.pt/~jja/updates/lups.html`

PolicyUpdater [Crescini and Zhang, 2005]

- Section 4.4
- `www.cit.uws.edu.au/~jcrescin/projects/PolicyUpdater/`

Appendix C

Stable-Models Procedure

This is a brief informal process from the literature on the procedure to compute *stable models* by means of a *reduct* and *Herbrand model* in a procedural approach. Note the alternative to Herbrand model in this case is the *minimal closure* of the reduct.

$\Pi_{\mathcal{M}}$: given a set of atoms $\mathcal{M}$ from Π. The *reduct program* $\Pi_{\mathcal{M}}$ is constructed by deleting:

(a) each rule ρ with a negative literal $\neg\ell \mid \ell \in \mathcal{M}$.

(b) all negative literals of the remaining rules.

$\mathcal{M}_H$: *Herbrand* (Unique) *model* of $\Pi_{\mathcal{M}}$. If coincides with $\mathcal{M}$, $\mathcal{M}$ is a *stable model* of Π.

Cn: **alternative** to $\mathcal{M}_H$: *minimal closure of* $\Pi_{\mathcal{M}}$ or $\mathsf{Cn}(\Pi_{\mathcal{M}})$:

- for a relation R, a class A is said to be closed under R, if whenever $x \in A$ and xRy then $y \in A$ (i.e., $R[A] \subseteq A$).
- *minimality*: no proper subset of $\Pi_{\mathcal{M}}$ is closed under $\Pi_{\mathcal{M}}$ and consistent.

SM: Any stable set (model) of Π is a minimal Herbrand model (*minimal closure*) of Π.

Glossary

$(\Pi, \mathcal{N}, <)$ Prioritised Logic Program. 67

$H^{\Pi}(\mathcal{S})$ Weak-constraint objective function. 34, 128, 157

$\mathsf{Head}(\rho) \leftarrow \mathsf{Body}(\rho) \cup \{\neg\alpha\}$ ρ-relaxed form. 111, 141

$\equiv$ Equivalence. 16, 17, 118

$\equiv_{\mathcal{H}_{\mathsf{I}}}$ Equivalence in $\mathcal{H}_{\mathsf{I}}$-logic. 16

$\equiv_{\mathsf{ASP}}$ Equivalence in ASP. 118

$\mathcal{M}(\Delta)$ Generalised Answer Set. 39

$\mathcal{M}_H$ Herbrand Model. 191, 193

$\mathsf{Name}(\cdot,\cdot)$ Injective naming function that assigns to each rule in a program a unique name [Eiter et al., 2002]. 44

$N_i^{\Pi}(\mathcal{S})$ Weak constraints at level i violated by $\mathcal{S}$. 34, 128, 157

$\Pi_{\mathcal{A}^*} = \langle\, \Pi, \mathcal{A}^* \,\rangle$ Abductive Logic Program. 112, 117, 121, 122, 124, 141, 146, 147, 149, 150, 193

:– `Prolog` derivation; ASP derivation. 33

$\mathsf{S}_{\mathsf{PLP}}(\mathsf{U}_{\mathsf{PLP}}(\mathcal{B}, \Pi))$ Set of all *resulting knowledge bases* of $\mathsf{U}_{\mathsf{PLP}}(\mathcal{B},\Pi)$ [Zhang, 2006]. 73

$\mathfrak{L}_{\Pi}$ Signature of Π. 29, 30, 32, 39, 100, 105, 112, 120, 146, 153, 158–160, 193

$\mathfrak{L}_{N_1^{\Pi}(\mathcal{M})}$ Signature of $N_1^{\Pi}(\mathcal{M})$. 158

$\mathfrak{L}_{\langle\, \Pi,\mathcal{A}^* \,\rangle}$ Signature of an abductive program $\langle\, \Pi, \mathcal{A}^* \,\rangle$. 39, 158

$\mathfrak{L}_{\Omega(\Pi)}$ Signature of $\Omega(\Pi)$. 158

:$\sim$ Weak-constraint derivation. 33

:$\sim \ell[w : l]$ Simple Weak constraint. 158

$\mathcal{W}(\Pi, \mathcal{A}^*)$ Translation into a weak-constraint program —ω-program. 158

$\mathbf{Cn}(\cdot)$ Consequence operation. 20

$\mathbf{Cn}(\Pi)$ Minimal closure of Π. 29

$\overline{\mathcal{M}} = \mathcal{A} \setminus \mathcal{M} \mid \mathcal{M} \subseteq \mathcal{A}$
$\mathcal{M}$-complement. 29

$\overline{a}$ Newly introduced atom uniquely associated with a. 56

$\equiv_{\mathcal{S}}$ Relaxed-sequence Equivalence. 112, 113, 124, 146, 194

$\equiv_{\mathcal{N}_2}$ Equivalence in $\mathcal{N}_2$—strongly equivalence. 32

$>_c$ Cardinality-preference Order. 37, 103, 194

$>_i$ Inclusion-preference Order. 37, 103, 194

$>_k$ $\times$-preference Order. 37, 103, 194

$l^{\Pi}_{\mathbf{max}}$ Maximum level over the weak constraints in Π. 34

$\leq_{\mathcal{A}^*}$ Abductive Inclusion Order. 39, 194

$\leq_{\mathcal{S}}$ Relaxed-sequence Order. 112–114, 120, 124, 146, 159, 194, 200

$\not\leq_{\mathcal{A}^*}$ Abductive Inclusion Order. 39, 194

new-ℓ Extending propositional literal to ℓ [Zhang, 2006]. 71

$\langle\, \Pi, \mathcal{A}^* \,\rangle$ Abductive Logic Program. 39

$\langle\, \Pi, \mathcal{A}^\bullet \,\rangle$ Extended Abductive Logic Program. 54

$\Pi_{\mathcal{M}}$ *Reduct program of* Π. 191, 194, 205

$\mathcal{P}^<$ *Reduct program of* $\mathcal{P}$ with respect to "$<$" [Zhang, 2006]. 68, 69

$\mathbf{rej}(\rho)$ New pair-wise unique atom. 44

$\Pi' \cup \Pi_2 \cup \Pi_G$ $\otimes'_o$-update program. 147

$\Pi' \cup \Pi_2 \cup \Pi_G$ $\otimes_o$-update program. 141

$\mathbf{U_{PLP}}(\mathcal{B}, \Pi) = (\Pi^*, \mathcal{N}, <)$ *Specification of updating* $\mathcal{B}$ *with* Π is a PLP over $\mathcal{L}_{\mathsf{new}}$ [Zhang, 2006]. 71, 73

$\vdash$ Derivation —inference. 20

$w^{\Pi}_{\mathbf{max}}$ Maximum weight over the weak constraints in Π. 34

$\Omega(\Pi)$ Weak constraints occurring in Π. 33, 34, 127, 128, 156–158

$weight(w)$ Weight of weak constraint w. 34, 128, 157

$e(\Pi, \mathcal{S})$ e-program [Zhang, 2006]. 67

f_Π Auxiliary function that maps levelled weights to weights without levels. 34, 128, 157

(Auxiliary symbol for grouping. 28

) Auxiliary symbol for grouping. 28

– Alternative strong-negation symbol. 28

$\mathsf{Def}(\mathcal{P}, \mathcal{M}) = \{\neg a \mid \nexists \rho \in \rho(\mathcal{P}), \mathsf{Head}(\rho) = a, \mathcal{M} \models \mathsf{Body}(\rho)\}$. 51

$\supset$ Classical implication. 29

$\leftarrow$ Derivation. 28, 29

$\mathsf{Rej}(\mathcal{P}, \mathcal{M}) = \{\rho \mid \rho \in \Pi_i, \exists \rho' \in \Pi_j, i \leq j, \rho \bowtie \rho', \mathcal{M} \models \mathsf{Body}(\rho')\}$. 51

$\bot$ Falsum. 28

$\neg$ Default negation symbol. 18, 19, 28

$\Pi^{\mathcal{S}} = \mathsf{Mon}(\{\rho \mid \{q_{m+1}, \ldots, q_n\} \cap \mathcal{S} = \emptyset\})$
Answer-sets *reduct program*. 29

$\top$ Verum. 28

, Connective. 28

; Meta-connective. 28

$\mathcal{L}_{\mathsf{ASP}}$ Language of Answer Sets. 28, 195

$\mathcal{B}$ *initial consistent knowledge base* of ground literals of a language $\mathcal{L}$ [Zhang, 2006]. 71

$\mathcal{B}$**'** *possible resulting knowledge base* with respect to $\mathsf{U}_{\mathsf{PLP}}(\mathcal{B},\Pi)$ [Zhang, 2006]. 73

$Bel(\mathcal{E})$ Belief set of $\mathcal{E}$: $Bel(\mathcal{E}) = \{\psi \mid \phi \vdash \psi\}$. 22

$\mathcal{E}$ Epistemic state. 22, 24

$\mathcal{K}$ Belief/Knowledge base. 21, 22

$\overline{\mathcal{M}} = \mathsf{least}(\rho(\mathcal{P}) \setminus \mathsf{Rej}(\mathcal{P}, \mathcal{M}) \cup \mathsf{Def}(\mathcal{P}, \mathcal{M}))$. 51

$\oplus_1$ $\mathcal{N}_2$-approach for updates. 83, 195

$\oplus_2$ $\mathcal{N}_2$-approach for updates. 81–83, 184–186, 196

$\ominus$ Expansion operator. 20

$|$ Alternative disjunction symbol. 28

$\oplus_R$ "Refined" DyLP operator. 51

$\uplus$ Expansion operator. 20, 21

ℓ Literal. 28, 29, 33, 34, 44, 45, 47, 49, 67, 68, 71–73, 104, 118, 121, 127, 141, 148, 156–158, 191, 193, 194, 196, 207

Mon Monotonic counterpart. 29

$\mathcal{M}^i(\Pi)$ set of rules of degree i satisfied by $\mathcal{M}$. 37, 102

$\mathcal{N}$ Naming function, mapping each rule in Π to a name [Zhang, 2006]. 67, 72, 206

$\mathcal{L}_{\mathbf{new}}$ extension to $\mathcal{L}$, by propositional literals of the form $\mathsf{new}\text{-}\ell \mid \ell \in \mathcal{L}$ [Zhang, 2006]. 71, 194, 196

$\otimes_o$ Update operator. 141

$\otimes'_o$ Update operator. 147

$\oslash$ Update operator. 97, 145

$\Pi_{(\Pi_0,\Pi_1)}$ Transformed program from Π_0 with respect to Π_1 such that $\Pi_{(\Pi_0,\Pi_1)}$ is a maximal subset of Π_0 such that $\mathcal{S}_{(\Pi_0,\Pi_1)}$ is coherent with $\Pi_{(\Pi_0,\Pi_1)}$ and $\mathcal{S}_{(\Pi_0,\Pi_1)} \in \mathsf{S}_{\mathsf{PLP}}(\mathsf{U}_{\mathsf{PLP}}(\mathcal{S}_{\Pi_0}, \Pi_1))$ [Zhang, 2006]. 74

Π'_0 Possible resulting program of $\mathsf{U}_{\diamond_\mathsf{Z}}(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$ after updating Π_0 with Π_1 [Zhang, 2006]. 78

Π^* Logic Program consisting of initial-knowledge rules, inertia rules and update rules [Zhang, 2006]. 71

$\mathsf{U}_{\diamond_\mathsf{Z}}(\Pi_0, \Pi_1)$ PLP Specification $(\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$ of updating Π_0 with Π_1 [Zhang, 2006]. 75

$*$ Belief-base revision operator. 21

$\dot{+}$ Belief-set revision operator. 20

$\circ$ Belief-state revision operator. 22, 151

Π' Relaxed program. 111, 145

$\otimes$ Update-sequence operator. 114, 146

$\otimes'$ Update-sequence operator. 124

$\sim$ Strong-negation symbol. 18, 19, 28

$\mathcal{S}_{\Pi_0}$ answer set of Π_0 [Zhang, 2006]. 74

$\mathcal{S}_{(\Pi_0,\Pi_1)}$ answer set of the update of $\mathcal{S}_{\Pi_0}$ with Π_1 [Zhang, 2006]. 74

$\circledast$ Belief-base update operator. 23

$\circledast$ Epistemic-state update operator. 24

$\Pi_1 \otimes \Pi_2 \otimes \cdots \otimes \Pi_n$ $\otimes$-update sequence. 114, 119, 146

$\Pi_1 \otimes' \Pi_2 \otimes' \cdots \otimes' \Pi_n$ $\otimes'$-update sequence. 44, 124

$\Pi_\times^{\mathcal{M}}$ *Reduct program of* Π with respect to $\mathcal{M}$ [Brewka et al., 2004]. 36, 102

$\rho_\times^{\mathcal{M}}$ *Reduct of* ρ with respect to $\mathcal{M}$ [Brewka et al., 2004]. 36, 101

Index

Vita

Juan Carlos Acosta Guadarrama was born in Toluca, Mexico, and lived in several cities until moving to Clausthal in 2005. He received a B.Sc. degree in Computing Systems Engineering from University of the Americas, Puebla and a M.Sc. degree specialised in Computing Systems from the same university. He has been employed in industry, institutions and services, as a unix-server administrator, webmaster, Internet distributed systems and security, staff training, tech support, computing engineering and teacher assistant. During his university studies, he got a bronze medal in a national tournament of tae kwon do, organized by Moo Duk Kwan. From 2005 onwards, he has been enrolled in the doctoral program in Computer Science at TU-Clausthal.